

**A Combined Simulation and Optimization Based Method for  
Predictive - Reactive Scheduling of Flexible Production  
Systems Subject to Execution Exceptions**

Dissertation  
for the partial fulfillment of the requirements of the degree  
DOCTOR OF BUSINESS ECONOMICS  
(Dr. rer. pol.)  
of the Heinz Nixdorf Institute, University of Paderborn, Germany

Submitted by  
M.Sc. Kiran R Mahajan  
33102 Paderborn

Paderborn, March 2007

**Rector: Prof. Dr. Peter F. E. Sloane**  
**Referee: Prof. Dr. -Ing. habil. Wilhelm Dangelmaier**  
**Co-Referee: Prof. Dr. rer. nat. Leena Suhl**  
**Co-Referee: Prof. Dr. math. Friedhelm Meyer auf der Heide**

I dedicate this thesis to aayi, papa,  
asawari, pitu and saylee

## I SUMMARY

Today, in order to gain competitive advantage, manufacturers around the world make a combined use of innovative production technologies and production processes and methods. Technologies like optimization and discrete-event simulation are used to test the effects of alternative policies, which help to organize manufacturing operations. However, an integrated platform for simulation and optimization algorithms within which the user could generate and control the effects of the alternative policies is missing. Alternative control policies become even more relevant today as there is a need to keep complex manufacturing systems stable on the occurrence of execution exceptions and at the same time meeting optimization goals.

The work discussed in this report complements the work in the areas of combination of simulation and optimization technologies. The system developed is a predictive-reactive system which combines both the technologies. The predictive part determines the feasible schedule to be used for a parallel machines flow shop (PMFS) – a flexible manufacturing system, in a predictive way and serves as a starting point for the analysis carried out later. It considers a mix of fixed and flexible part flows, delivery constraints, buffer constraints, part flow constraints, and optimization constraints during its computations. This schedule is generated using a combination of rule-based simulation and optimization: using first the optimization algorithm to compute a rough plan, followed by using a rule based simulation system to locally fine tune the plan, and obtain the final schedule. The schedule generated by this predictive system, when implemented in the real world system is adapted by the reactive part of the system by generating alternative policies on the occurrence of system exceptions. These new alternative policies constitute the significant processes in the real world for a corresponding exception. In the reactive phase, alternative policies too are generated using a combination of simulation and optimization. The optimization algorithm brings the deviation from the predictive schedule back to its original trajectory as much as possible. In other words, it tries to change as less as possible. It does this while considering and solving adaptation synchrony problems (the problem that computations and changes in the real-world take time, while the real-world continues to evolve leading to differing states being used at different times) that may occur on the shop floor due to the change (or the new schedule). The simulation based system also predicts if there will be problems in the near future due to the rescheduling action and tries to generate solutions based on rules, which make sure that the future execution of the schedule in the real-world will be problem free. The final rescheduling solution is evaluated by the simulation system and then implemented in the real-world.

The predictive system is tested using several test configurations for the

effectiveness of the rule-based simulation system and the optimization algorithm to meet optimization criteria and goals. For the reactive system, a quantitative analysis is made considering system performance, the characteristics of the exceptions, and the ability to meet the rescheduling aims.

Results obtained show that the predictive scheduling method of combining rule-based simulation and optimization is promising as it provides unique insights on further improvement of performance measures of job finishing times, makespan and delivery times. The simulation assisted rescheduling system also resulted in additional performance measures of controlling deviations, which are well researched in this thesis as well as evaluating the rescheduling solution and solving adaptation synchrony problems. Definite quantitative information on bringing the schedule back to its original trajectory as much as possible, its limits, and at the same time predicting effects of current changes and solving them before hand were obtained. It is seen that the methods developed are effective. The overall approach suggested in this report is based on the integration of technologies like optimization and discrete-event simulation, thus making it unique in the application of today's industrial problems.

## II CONTENTS

|   |            |
|---|------------|
| <b>Summary</b> .....  | <b>i</b>   |
| <b>Contents</b> .....   | <b>ii</b>  |
| <b>Notations</b> .....  | <b>iii</b> |
| <b>Figures</b> .....  | <b>iv</b>  |
| <b>Tables</b> .....   | <b>v</b>   |
| <br>  |            |
| <b>1. Introduction</b> .....  | <b>1</b>   |
| <br>  |            |
| <b>2. Problem areas addressed</b> .....   | <b>2</b>   |
| 2.1 Introduction.....   | 2          |
| 2.2 Scheduling and re-scheduling of flexible production systems subject to execution exceptions.....    | 2          |
| 2.2.1 Predictive scheduling of flexible production systems subject to known and unknown exceptions..... | 2          |
| 2.2.1.1 Approaches to predictive scheduling – combining simulation and optimization.....                | 2          |
| 2.2.1.2 Fixed and flexible material flow routings with and without delivery time constraints.....       | 3          |
| 2.2.1.3 Inclusion of known and unknown events and exceptions...3  |            |
| 2.2.2 Reactive scheduling of flexible production systems subject to known and unknown exceptions.....   | 3          |
| 2.2.2.1 Approach to rescheduling – combination of simulation and optimization.....                      | 4          |
| 2.2.2.2 Approach to rescheduling – when to reschedule.....  | 4          |
| 2.2.2.2.1 Real-time control issues in rescheduling – Adaptation Synchrony.....                          | 4          |
| 2.2.2.3 Rescheduling method.....  | 5          |
| 2.2.2.3.1 Deviation match-up with predictive schedule.....  | 5          |
| 2.2.2.3.2 Rescheduling as late as possible.....   | 7          |
| 2.2.2.4 Estimating the impact of the rescheduling step on predictive schedule.....                      | 8          |
| 2.3 Optimization objectives.....  | 8          |
| 2.4 Assumptions used in the study.....  | 9          |

|   |           |
|---|-----------|
| 2.5 Real Time Control terms used in the study.....  | 10        |
| 2.6 Structure of the report.....  | 10        |
| <b>3. State of the art.....</b>   | <b>12</b> |
| 3.1 Introduction.....   | 12        |
| 3.2 Predictive scheduling of flexible production systems.....                               | 12        |
| 3.2.1 Various approaches of predictive scheduling.....                                      | 12        |
| 3.2.1.1 Optimal-Analytical approaches of scheduling the PMFS<br>problem.....                | 12        |
| 3.2.1.2 Simulation based approaches of scheduling.....                                      | 18        |
| 3.2.1.2.1 <i>AutoSched</i> : Simulation based scheduler.....                                | 18        |
| 3.2.1.2.2 <i>SIMUL8-Planner</i> : Simulation based planning<br>and scheduling.....          | 19        |
| 3.2.1.2.3 <i>ISSOP</i> Simulation based scheduling.....                                     | 21        |
| 3.2.1.2.4 <i>Preactor International</i> case of simulation<br>based scheduling.....         | 22        |
| 3.3 Reactive scheduling of flexible production systems.....                                 | 23        |
| 3.3.1 Various approaches of reactive scheduling.....  | 23        |
| 3.3.1.1 Simulation based completely reactive approaches.....                                | 23        |
| 3.3.1.2 Predictive – reactive scheduling.....   | 26        |
| 3.3.1.2.1 When to re-schedule?.....   | 26        |
| 3.3.1.3 Predictive-reactive scheduling versus completely reactive<br>approaches.....        | 28        |
| 3.4 Conclusions.....  | 29        |
| <b>4. Work program and objectives.....</b>  | <b>31</b> |
| 4.1 Introduction.....   | 31        |
| 4.2 Work program.....   | 31        |
| 4.3 Conclusions.....  | 32        |
| <b>5. Concepts and solutions.....</b>   | <b>33</b> |
| 5.1 Introduction.....   | 33        |
| 5.2 Concept and solutions for predictive scheduling.....                                    | 33        |
| 5.2.1 Optimization algorithm.....   | 35        |
| 5.2.1.1 Initialization.....   | 35        |
| 5.2.1.2 Detailed procedure to handle deterministic events<br>and delivery constraints.....  | 39        |
| 5.2.2 Simulation based Flow Analyzer Module (FAM) for assessing<br>predictive schedule..... | 46        |
| 5.2.2.1 Flow Analyzer Module (FAM).....   | 48        |
| 5.2.2.1.1 Rule generation for handling optimality.....                                      | 49        |
| 5.2.2.1.2 Rule generation for handling validity by<br>avoiding bottlenecks.....             | 54        |

|           |  |           |
|-----------|--|-----------|
| 5.2.2.2   | Sequential rule firing and its consequences.....   | 58        |
| 5.3       | Concept and solutions for reactive scheduling.....   | 59        |
| 5.3.1     | Justifications for using these methods.....  | 64        |
| 5.3.2     | Matchup rescheduling approach for real-time control.....                                       | 64        |
| 5.3.2.1   | Adaptation Synchrony Analysis (ASA) during<br>rescheduling.....                                | 67        |
| 5.3.2.2   | Detail algorithm for match-up rescheduling and the<br>Adaptation Synchrony Analysis (ASA)..... | 72        |
| 5.3.2.3   | Post rescheduling analysis using the simulation<br>based Flow Analyzer Module (FAM).....       | 76        |
| 5.3.3     | The selective re-routing approach for real-time control.....                                   | 76        |
| 5.3.3.1   | Concept of selective re-routing.....   | 77        |
| 5.3.3.2   | Detail algorithm for selective re-routing.....   | 80        |
| 5.4       | Conclusions.....   | 82        |
| <b>6.</b> | <b>Integration and overall framework.....</b>  | <b>84</b> |
| 6.1       | Introduction.....  | 84        |
| 6.2       | Overall system framework.....  | 84        |
| 6.3       | Integration of the entire system.....  | 84        |
| 6.4       | Conclusions.....   | 86        |
| <b>7.</b> | <b>Prototype software realized.....</b>  | <b>87</b> |
| 7.1       | Introduction.....  | 87        |
| 7.2       | Simulation software <i>eM-Plant</i> .....  | 87        |
| 7.2.1     | Programming language SimTalk.....  | 89        |
| 7.2.2     | Important concepts of <i>eM-Plant</i> .....  | 90        |
| 7.3       | Structure of the implementation.....   | 91        |
| 7.4       | Application flow of the implemented system.....  | 93        |
| 7.5       | Predictive scheduling system.....  | 95        |
| 7.5.1     | Optimization algorithm based predictive scheduling.....  | 95        |
| 7.5.1.1   | Customization of <i>eM-Plant</i> .....   | 95        |
| 7.5.1.2   | Workflow of the whole system.....  | 96        |
| 7.5.2     | Simulation assisted FAM for predictive scheduling.....   | 99        |
| 7.5.2.1   | Customization of <i>eM-Plant</i> .....   | 99        |
| 7.5.2.2   | Workflow of the whole system.....  | 99        |
| 7.5.3     | Running the simulation and optimization based predictive<br>scheduling system.....             | 101       |
| 7.5.3.1   | Starting the system.....   | 101       |
| 7.6       | Reactive scheduling system.....  | 105       |
| 7.6.1     | Match-up rescheduling system.....  | 105       |
| 7.6.1.1   | Customization of <i>eM-Plant</i> .....   | 105       |
| 7.6.1.2   | Workflow of the whole system.....  | 106       |

|           |   |            |
|-----------|---|------------|
| 7.6.1.3   | Running the simulation and optimization based reactive scheduling system..... | 112        |
| 7.6.2     | Selective re-routing system.....  | 116        |
| 7.6.2.1   | Customization of <i>eM-Plant</i> .....  | 116        |
| 7.6.2.2   | Workflow of the whole system.....   | 116        |
| 7.6.2.3   | Running the simulation and the system.....                                    | 117        |
| 7.7       | Conclusions.....  | 117        |
| <b>8.</b> | <b>Quantitative assessment of approaches.....</b>                             | <b>118</b> |
| 8.1       | Introduction.....   | 118        |
| 8.2       | Testing predictive scheduling system: Parameters and test results....         | 118        |
| 8.2.1     | Computational times of the predictive scheduling system.....                  | 119        |
| 8.2.1.1   | Test 1 data, results and discussions.....                                     | 119        |
| 8.2.1.2   | Test 2 data, results and discussions.....                                     | 120        |
| 8.2.2     | Delivery time optimization results.....                                       | 122        |
| 8.2.2.1   | Test 3 data, results and discussions.....                                     | 122        |
| 8.2.2.2   | Test 4 data, results and discussions.....                                     | 125        |
| 8.2.3     | Test 5 data, results and discussions.....                                     | 129        |
| 8.2.3.1   | Makespan comparison for various methods.....                                  | 129        |
| 8.2.3.2   | Performance benefits of the simulation based FAM system.....                  | 130        |
| 8.2.4     | Test 6 data, results and discussions.....                                     | 131        |
| 8.2.4.1   | Makespan comparison for various methods.....                                  | 131        |
| 8.2.4.2   | Performance benefits of the simulation based FAM system.....                  | 131        |
| 8.2.5     | Test 7 data, results and discussions.....                                     | 132        |
| 8.2.5.1   | Makespan comparison for various methods.....                                  | 133        |
| 8.2.5.2   | Performance benefits of the simulation based FAM system.....                  | 133        |
| 8.2.6     | Test 8 data, results and discussions.....                                     | 134        |
| 8.2.6.1   | Makespan comparison for various methods.....                                  | 134        |
| 8.2.6.2   | Performance benefits of the simulation based FAM system.....                  | 134        |
| 8.2.7     | Test 9 data, results and discussions.....                                     | 135        |
| 8.2.7.1   | Makespan comparison for various methods.....                                  | 136        |
| 8.2.7.2   | Performance benefits of the simulation based FAM system.....                  | 136        |
| 8.2.8     | Test 10 data, results and discussions.....                                    | 137        |
| 8.2.8.1   | Makespan comparison for various methods.....                                  | 137        |
| 8.2.8.2   | Performance benefits of the simulation based FAM system.....                  | 137        |
| 8.3       | Testing reactive scheduling system: Parameters and test results.....          | 138        |
| 8.3.1     | Testing the simulation assisted match-up rescheduling system..                | 139        |



|               |  |            |
|---------------|--|------------|
| 8.3.1.1       | Validating the detailed working of the ASA.....  | 139        |
| 8.3.1.2       | Test 1 with ASA implemented in steps and with reactive FAM.....                            | 141        |
| 8.3.1.2.1     | Test 1a without ASA and with reactive FAM.....   | 141        |
| 8.3.1.2.2     | Test 1b with limited ASA but with reactive FAM.....  | 143        |
| 8.3.1.2.3     | Test 1c with full ASA and with reactive FAM....  | 144        |
| 8.3.1.3       | Test 2 with ASA implemented in steps to test effectiveness of the rescheduling system..... | 145        |
| 8.3.1.3.1     | Test 2a without ASA but with reactive FAM.....   | 145        |
| 8.3.1.3.2     | Test 2b with ASA and with reactive FAM.....  | 147        |
| 8.3.2         | Testing the selective re-routing system.....   | 148        |
| 8.3.2.1       | Test 1 to check late change criteria using the selective re-routing system.....            | 148        |
| 8.3.2.2       | Test 2 to check late change criteria using the selective re-routing system.....            | 149        |
| 8.4           | Conclusions.....   | 150        |
| <b>9.</b>     | <b>Contributions, conclusions and future research.....</b>                                 | <b>152</b> |
| 9.1           | Contributions.....   | 152        |
| 9.2           | Conclusions.....   | 153        |
| 9.3           | Future research.....   | 154        |
| <b>VI</b>     | <b>ACKNOWLEDGMENTS.....</b>  | <b>157</b> |
| <b>VII</b>    | <b>REFERENCES.....</b>   | <b>158</b> |
| <b>VIII</b>   | <b>APPENDICES.....</b>   | <b>162</b> |
| Appendix 1:   | Detail iterations for predictive scheduling algorithm and validation.....                  | 162        |
| Appendix 1.1: | Detail events and results for simulation based FAM and validation.....                     | 176        |
| Appendix 2:   | Detail iteration for matchup rescheduling algorithm and validation.....                    | 184        |
| Appendix 3:   | Detail iteration for selective rerouting rescheduling Algorithm and validation.....        | 186        |
| Appendix 4:   | Methods, Tables and Variable objects used during implementation.....                       | 188        |
| Appendix 5:   | Job finishing times for Test 5.....  | 199        |

Appendix 6: Job finishing times for Test 6.....200

### III NOTATIONS

#### General notations:

- $t$  = Time  
 $Sch_{predictive}$  = Predictive schedule  
 $Sch_{reactive}$  = Reactive schedule  
 $F_j$  = Flow time or make-span  
 $\bar{F}$  = Mean flow time  
 $\bar{J}$  = Average in-process inventory levels  
 $C_j$  = Completion time for job  $j$   
 $r_j$  = Time job  $j$  arrived in the system  
 $t_D$  = Time when disturbance happened  
 $t_{Dur}$  = Duration of disturbance  
 $makespan^{UB}$  = Makespan upper bound  
 $devSq$  = Sequence deviation  
 $devSt$  = Starting time deviation  
 $it$  = Iteration number

#### Notations for stages and machines:

- $J$  = Number of stages in the flowshop  
 $m_j$  = Number of machines at stage  $-j$   
 $M_j$  =  $\{1, 2, \dots, m_j\}$ : Set of machines at stage  $-j$   
 $a_{j,k}^m$  = Ready / available time for machine  $-k$  at stage  $-j$   
 $M'_j \subseteq M_j$  = Set of machines at stage  $-j$  available at time  $t$  (that is  $a_{j,k}^m \leq t$ )  
 $k_j^m$  = Machine  $k$ , with id  $m$ , maintained at stage  $j$   
 $k_{tSt}$  = Time when disturbance starts on machine  $k$   
 $k_{tEnd}$  = Time when disturbance ends on machine  $k$

|              |   |
|--------------|---|
| $tl_{tAv}^i$ | = Tool availability time for job $i$                    |
| $tl_{tNum}$  | = Number of tools for which availability is set         |
| $i_t$        | = Material availability time for job $i$                |
| $Bf_c$       | = Buffer capacity                                       |
| $Bf_{cc}$    | = Current\Remaining buffer capacity                     |
| $Bf_{CN}$    | = Current number of jobs in buffer                      |
| $m'$         | = Selected machine                                      |
| $a_{BN,k}^m$ | = Ready time for machine $k$ on critical stage          |
| $s_{i,BN}$   | = Earliest start time for job $i$ on the critical stage |

#### Notations for jobs:

|                   |   |
|-------------------|---|
| $i_{Sp}$          | = Special job   |
| $i_c$             | = Job currently processed on a machine                                      |
| $R$               | = Special job routing in system   |
| $I$               | = $\{1,2,\dots,n\}$ : set of jobs to be scheduled                           |
| $I_0 \subseteq I$ | = Set of jobs already scheduled   |
| $I' = I - I_0$    | = Set of jobs not scheduled yet   |
| $a_{i,j}$         | = Time when job – $i$ becomes available at stage – $j$ (“ready time”)       |
| $p_{i,j}$         | = Processing time for job – $i$ at stage – $j$                              |
| $q_{i,j}$         | = Work remaining (“tail”) for job – $i$ at stage – $j$                      |
| $s_{i,j}$         | = Earliest start time for job – $i$ at stage – $j$                          |
| $p_{i,j}^{Bf}$    | = Processing time for job – $i$ which is currently in buffer on stage – $j$ |
| $bd'$             | = Set of jobs on machine $k$ on stage $j$                                   |
| $bd''$            | = Set of jobs selected for rescheduling                                     |
| $d_t$             | = Delivery time   |
| $I''$             | = Jobs not yet scheduled from the list of standard jobs                     |
| $t_m$             | = Time machine becomes free after processing its current job – $i$          |

## IV FIGURES

|             |   |    |
|-------------|---|----|
| Figure 2.1  | Real-world state evolution and interaction with the rescheduling system.....                    | 5  |
| Figure 2.2  | Starting time deviation ( <i>devSt</i> ).....   | 6  |
| Figure 2.3  | Sequence deviation ( <i>devSq</i> ).....  | 7  |
| Figure 3.1  | A disjunctive graph formation for the PMFS problem.....   | 13 |
| Figure 3.2  | Highest lower bound stage selected as critical stage.....                                       | 16 |
| Figure 3.3  | Interaction of simulation and optimization tool for car painting.....                           | 22 |
| Figure 3.4  | General scheme of simulation based reactive scheduling.....                                     | 24 |
| Figure 5.1  | Overview of the predictive scheduling system.....   | 34 |
| Figure 5.2  | Inputs and outputs to the predictive algorithm.....   | 35 |
| Figure 5.3  | Accommodating delivery constraints for special jobs.....  | 43 |
| Figure 5.4  | Simulation based Flow Analyzer (FA) to achieve validity.....                                    | 47 |
| Figure 5.5  | Concept of the simulation and rule based Flow Analyzer.....                                     | 48 |
| Figure 5.6  | Dynamics of optimality and validity rule generation.....  | 51 |
| Figure 5.7  | Sequential rule generation.....   | 58 |
| Figure 5.8  | Consequences of sequential rule generation.....   | 59 |
| Figure 5.9  | Simulation and Optimization assisted reactive scheduling.....                                   | 60 |
| Figure 5.10 | Solution spaces and bounds.....   | 62 |
| Figure 5.11 | Time line of computations for simulation assisted reactive scheduling.....                      | 63 |
| Figure 5.12 | Matchup rescheduling for real-time control.....   | 65 |
| Figure 5.13 | Tree of iterations for positions and job rescheduling.....                                      | 67 |
| Figure 5.14 | Real-world state evolution and interaction with the rescheduling system.....                    | 68 |
| Figure 5.15 | Change charts filled up in the previous shift used in the current shift adaptation process..... | 69 |
| Figure 5.16 | Adaptation synchrony analysis computation.....  | 70 |
| Figure 5.17 | Earliest available machine on exception occurred stage.....                                     | 73 |
| Figure 5.18 | Determining the set of jobs to reschedule.....  | 73 |
| Figure 5.19 | Post rescheduling analysis using simulation based FAM.....                                      | 77 |
| Figure 5.20 | Selective rerouting for reactive scheduling (one iteration).....                                | 79 |
| Figure 6.1  | Overview of the simulation assisted production scheduling and rescheduling system.....          | 85 |
| Figure 6.2  | Building block of the total scheduling and rescheduling system.....                             | 86 |
| Figure 7.1  | Top frame of the developed system.....  | 92 |
| Figure 7.2  | Table frame which part of top frame.....  | 93 |
| Figure 7.3  | Application flow of the system.....   | 94 |

|  |     |
|--|-----|
| Figure 7.4 Data flow chart of the predictive system.....   | 98  |
| Figure 7.5 Data flow chart of the simulation assisted FAM for predictive scheduling.....               | 100 |
| Figure 7.6 Screenshot scheduling dialog.....   | 101 |
| Figure 7.7 Screenshot editing job processing times.....  | 102 |
| Figure 7.8 Screenshot editing job delivery times and routing constraints.....                          | 102 |
| Figure 7.9 Screenshot editing tool or resource availability.....                                       | 103 |
| Figure 7.10 Screenshot editing material availability.....  | 103 |
| Figure 7.11 Screenshot editing equipment availability.....   | 104 |
| Figure 7.12 Screenshot editing decision points, rule generators and conditions...                      | 105 |
| Figure 7.13 Data flow chart of the optimization and simulation based match-up rescheduling system..... | 107 |
| Figure 7.14 Detail working of the job shifting procedure.....  | 109 |
| Figure 7.15 Interaction of the ASA module with the rescheduling system.....                            | 110 |
| Figure 7.16 Continuation of the simulation to conduct reactive FAM analysis.....                       | 111 |
| Figure 7.17 Screenshot user notification of exception.....   | 113 |
| Figure 7.18 Screenshot rescheduling dialog.....  | 113 |
| Figure 7.19 Screenshot final results window.....   | 114 |
| Figure 7.20 Screenshot of the detailed rescheduling results.....                                       | 114 |
| Figure 7.21 Screenshot for change chart – transportation factor.....                                   | 115 |
| Figure 7.22 Screenshot for change chart – job set out time factor.....                                 | 115 |
| Figure 7.23 Screenshot for change chart – job set in time factor.....                                  | 115 |
| Figure 8.1 Computational times using the predictive scheduling system: Test 1..                        | 120 |
| Figure 8.2 Computational times using the predictive scheduling system: Test 2..                        | 121 |
| Figure 8.3 Setting delivery times and standard and special job flows.....                              | 122 |
| Figure 8.4 Setting machine maintenance times.....  | 123 |
| Figure 8.5 Setting material availability.....  | 123 |
| Figure 8.6 Resulting plan and schedules obtained with both systems.....                                | 124 |
| Figure 8.7 Job delivery table calculated by algorithm.....   | 124 |
| Figure 8.8 Makespan comparing pure optimization and optimization with FAM schedules.....               | 125 |
| Figure 8.9 Delivery constraints and job paths set up by user for special jobs.....                     | 126 |
| Figure 8.10a Resulting schedule obtained with optimization and simulation based FAM: Part 1.....       | 127 |
| Figure 8.10b Resulting schedule obtained with optimization and simulation based FAM: Part 2.....       | 128 |
| Figure 8.11 Makespan results for different methods of scheduling: Test 5.....                          | 130 |
| Figure 8.12 Percentage reduction in JFT using simulation based FAM: Test 5...                          | 130 |

|             |   |     |
|-------------|---|-----|
| Figure 8.13 | Makespan results for different methods of scheduling: Test 6.....                       | 132 |
| Figure 8.14 | Percentage reduction in JFT using simulation based FAM: Test 6....                      | 132 |
| Figure 8.15 | Makespan results for different methods of scheduling: Test 7.....                       | 133 |
| Figure 8.16 | Percentage reduction in JFT using simulation based FAM: Test 7....                      | 134 |
| Figure 8.17 | Makespan results for different methods of scheduling: Test 8.....                       | 135 |
| Figure 8.18 | Percentage reduction in JFT using simulation based FAM: Test 8....                      | 135 |
| Figure 8.19 | Makespan results for different methods of scheduling: Test 9.....                       | 136 |
| Figure 8.20 | Percentage reduction in JFT using simulation based FAM: Test 9....                      | 137 |
| Figure 8.21 | Makespan results for different methods of scheduling: Test 10.....                      | 138 |
| Figure 8.22 | Percentage reduction in JFT using simulation based FAM: Test 10..                       | 138 |
| Figure 8.23 | Ouput console window of eM-Plant simulation software.....                               | 140 |
| Figure 8.24 | Comparison of upper bounds and predictive schedule.....                                 | 142 |
| Figure 8.25 | Comparison of rescheduling result with predictive schedule.....                         | 142 |
| Figure 8.26 | Comparison of the rescheduling result with predictive schedule.....                     | 143 |
| Figure 8.27 | Comparison of upper bounds with predictive schedule.....                                | 145 |
| Figure 8.28 | Comparison of rescheduling system with predictive schedule.....                         | 145 |
| Figure 8.29 | Comparison of the rescheduling result with the predictive schedule..                    | 146 |
| Figure 8.30 | Comparison of the upper bounds with predictive schedule.....                            | 146 |
| Figure 8.31 | Comparison of rescheduling result with predictive shedule.....                          | 147 |
| Figure 8.32 | Job finishing times and events in the reactive system.....                              | 149 |
| Figure 8.33 | Job finishing times and events in the reactive system.....                              | 150 |
| Figure A1   | Screen shot of <i>eM-Plant</i> showing selections and lower bound<br>Calculations.....  | 175 |
| Figure A2   | Run-time results of the simulation based FAM in <i>eM-Plant</i><br>console: Part 1..... | 181 |
| Figure A3   | Run-time results of the simulation based FAM in <i>eM-Plant</i><br>console: Part 2..... | 182 |
| Figure A4   | Run-time results of the simulation based FAM in <i>eM-Plant</i><br>console: Part 3..... | 183 |
| Figure A5   | Predictive FAM schedule gantt chart.....  | 184 |
| Figure A6   | Detailed iterations and selections in <i>eM-Plant</i> console.....                      | 184 |
| Figure A7   | Results from the software run.....  | 185 |
| Figure A8   | Predictive FAM schedule gantt chart.....  | 186 |
| Figure A9   | Detailed iterations and selections in <i>eM-Plant</i> console.....                      | 186 |
| Figure A10  | Results from the software run.....  | 187 |
| Figure A11  | Gantt chart of rescheduling solution.....   | 187 |
| Figure A12  | Analysis of job finishing times for different methods of scheduling:<br>Test 5.....     | 199 |

Figure A13 Analyzing job finishing times for different scheduling methods:  
Test 6.....200



## V TABLES

|            |   |     |
|------------|---|-----|
| Table 3.1  | Comparison of heuristic with other approaches.....                      | 18  |
| Table 5.1  | Example data to explain algorithm.....                                  | 36  |
| Table 5.2  | List 1 of special unscheduled jobs with routing and delivery times..... | 37  |
| Table 5.3  | List 2 of special unscheduled jobs with only delivery times.....        | 38  |
| Table 5.4  | List 3 of special jobs with routings only.....                          | 38  |
| Table 5.5  | List 4 of standard jobs only.....                                       | 38  |
| Table 5.6  | Job ready times at stage 1.....   | 38  |
| Table 5.7  | Tails for each job at each stage.....                                   | 39  |
| Table 5.8  | Earliest start times at stage 1 for all jobs.....                       | 39  |
| Table 5.9  | Earliest start times at stage 2 for all jobs.....                       | 40  |
| Table 5.10 | Lower bounds computed for both stages.....                              | 40  |
| Table 5.11 | Updated machine ready times after scheduling job 7.....                 | 41  |
| Table 5.12 | Updated machine ready times after scheduling job 3.....                 | 45  |
| Table 5.13 | Interrelationships between decision points and rule generator.....      | 49  |
| Table 5.14 | Example jobs and system size.....                                       | 50  |
| Table 5.15 | Buffer capacities on stage 2.....                                       | 50  |
| Table 5.16 | Job routing according to optimization algorithm.....                    | 50  |
| Table 5.17 | Earliest time job $i$ will finish on stage.....                         | 53  |
| Table 5.18 | Job routing obtained after running simulation based FAM.....            | 53  |
| Table 5.19 | Example jobs and system size.....                                       | 54  |
| Table 5.20 | Buffer capacities on stage 2.....                                       | 54  |
| Table 5.21 | Job routing according to optimization algorithm.....                    | 54  |
| Table 5.22 | Earliest time job $i$ will finish on stage.....                         | 57  |
| Table 5.23 | Job routing obtained after running simulation based FAM.....            | 58  |
| Table 5.24 | Example jobs and system size.....                                       | 72  |
| Table 5.25 | Iterations and jobs selected for rescheduling.....                      | 74  |
| Table 5.26 | Capacities on earliest available alternative machine.....               | 74  |
| Table 5.27 | Example to explain rescheduling system.....                             | 80  |
| Table 5.28 | Iterations and jobs selected for rescheduling.....                      | 81  |
| Table 5.29 | Capacities on earliest available alternative machine.....               | 82  |
| Table 8.1  | Test parameters.....  | 119 |
| Table 8.2  | Test plan and relation to parameters.....                               | 119 |
| Table 8.3  | Data for test case 1.....   | 120 |
| Table 8.4  | Data for test case 2.....   | 121 |
| Table 8.5  | Data for test case 3.....   | 122 |
| Table 8.6  | Lateness measurements $L_j$ for Test 3.....                             | 125 |
| Table 8.7  | Data for test case 4.....   | 126 |

|            |   |     |
|------------|---|-----|
| Table 8.8  | Lateness measurements $L_j$ for Test 4.....                                   | 129 |
| Table 8.9  | Data for test case 5.....   | 129 |
| Table 8.10 | Data for test case 6.....   | 131 |
| Table 8.11 | Data for test case 7.....   | 133 |
| Table 8.12 | Data for test case 8.....   | 134 |
| Table 8.13 | Data for test case 9.....   | 136 |
| Table 8.14 | Data for test case 10.....  | 137 |
| Table 8.15 | Test plan for match-up rescheduling system.....                               | 139 |
| Table 8.16 | Data for testing working of ASA.....  | 140 |
| Table 8.17 | Data for test case 1.....   | 141 |
| Table 8.18 | Summary of results for Test 1.....  | 142 |
| Table 8.19 | Summary of results for Test 1b.....   | 143 |
| Table 8.20 | Settings used for Test 1c.....  | 144 |
| Table 8.21 | Summary of results for Test 1c.....   | 144 |
| Table 8.22 | Data used for Test case 2a.....   | 145 |
| Table 8.23 | Summary of results for Test 2a.....   | 146 |
| Table 8.24 | Data used for testing system.....   | 147 |
| Table 8.25 | Summary of results for Test 2b.....   | 147 |
| Table 8.26 | Test plan for testing selective re-routing system.....                        | 148 |
| Table 8.27 | Summary of results for Test 1.....  | 149 |
| Table 8.28 | Summary of results of Test 2.....   | 149 |
| Table A1   | Job ready times at stage - 1 ( $a_{i,1}$ ).....                               | 162 |
| Table A2   | Tails for each job at each stage.....   | 162 |
| Table A3   | Earliest start times at stage 1 for all jobs.....                             | 162 |
| Table A4   | Earliest start times at stage 2 for all jobs.....                             | 163 |
| Table A5   | Lower bounds computed for both stages.....                                    | 163 |
| Table A6   | Jobs scheduled list after this iteration.....                                 | 163 |
| Table A7   | Updating machine ready times on stage 1 and stage 2 after<br>iteration 2..... | 164 |
| Table A8   | Job ready times at stage - 1 ( $a_{i,1}$ ).....                               | 164 |
| Table A9   | Tails for each job at each stage.....   | 164 |
| Table A10  | Earliest start times at stage 1 for all jobs.....                             | 165 |
| Table A11  | Earliest start times at stage 2 for all jobs.....                             | 165 |
| Table A12  | Lower bounds computed for both stages.....                                    | 165 |
| Table A13  | Jobs scheduled list after this iteration.....                                 | 165 |
| Table A14  | Updating machine ready times on stage 1 and stage 2 after<br>iteration 3..... | 166 |
| Table A15  | Job ready times at stage - 1 ( $a_{i,1}$ ).....                               | 166 |
| Table A16  | Tails for each job at each stage.....   | 166 |

|  |     |
|--|-----|
| Table A17 Updating machine ready times on stage 1 and stage 2 after iteration 4..... | 166 |
| Table A18 Earliest start times at stage 1 for all jobs.....                          | 167 |
| Table A19 Earliest start times at stage 2 for all jobs.....                          | 167 |
| Table A20 Lower bounds computed for both stages.....                                 | 167 |
| Table A21 Jobs scheduled list after this iteration.....                              | 167 |
| Table A22 Jobs at this iteration.....  | 169 |
| Table A23 Job ready times at stage - 1 ( $a_{i,1}$ ).....                            | 169 |
| Table A24 Earliest start times at stage 1 for all jobs.....                          | 169 |
| Table A25 Earliest start times at stage 2 for all jobs.....                          | 169 |
| Table A26 Updating machine ready times on stage 1 and stage 2 after iteration 5..... | 169 |
| Table A27 Jobs scheduled list.....   | 170 |
| Table A28 Jobs in this iteration.....  | 170 |
| Table A29 Job ready times at stage - 1 ( $a_{i,1}$ ).....                            | 170 |
| Table A30 Earliest start times at stage 1 for all jobs.....                          | 170 |
| Table A31 Earliest start times at stage 2 for all jobs.....                          | 170 |
| Table A32 Updating machine ready times on stage 1 and stage 2 after iteration 6..... | 171 |
| Table A33 Jobs scheduled list after this iteration.....                              | 171 |
| Table A34 Job ready times at stage - 1 ( $a_{i,1}$ ).....                            | 171 |
| Table A35 Earliest start times at stage 1 for all jobs.....                          | 171 |
| Table A36 Earliest start times at stage 2 for all jobs.....                          | 171 |
| Table A37 Updating machine ready times on stage 1 and stage 2 after iteration 7..... | 172 |
| Table A38 Jobs scheduled list after this iteration.....                              | 172 |
| Table A39 Jobs for this iteration.....   | 172 |
| Table A40 Job ready times at stage - 1 ( $a_{i,1}$ ).....                            | 172 |
| Table A41 Earliest start times at stage 1 for all jobs.....                          | 172 |
| Table A42 Earliest start times at stage 2 for all jobs.....                          | 173 |
| Table A43 Updating machine ready times on stage 1 and stage 2 after iteration 8..... | 173 |
| Table A44 Jobs scheduled list after this iteration.....                              | 173 |
| Table A45 Jobs in this iteration.....  | 173 |
| Table A46 Job ready times at stage - 1 ( $a_{i,1}$ ).....                            | 173 |
| Table A47 Earliest start times at stage 1 for all jobs.....                          | 174 |
| Table A48 Earliest start times at stage 2 for all jobs.....                          | 174 |
| Table A49 Updating machine ready times on stage 1 and stage 2 after iteration 9..... | 174 |

|  |     |
|--|-----|
| Table A50 Jobs scheduled list after this iteration.....                                  | 174 |
| Table A51 Updating machine ready times on stage 1 and stage 2 after<br>iteration 10..... | 174 |
| Table A52 Job exit times for all jobs.....   | 175 |
| Table A53 FAM analysis of job 3.....   | 176 |
| Table A54 FAM analysis of job 4.....   | 176 |
| Table A55 FAM analysis of job 1.....   | 177 |
| Table A56 FAM analysis of job 2.....   | 177 |
| Table A57 FAM analysis of job 3.....   | 178 |
| Table A58 FAM analysis of job 4.....   | 178 |
| Table A59 FAM analysis of job 1.....   | 178 |
| Table A60 FAM analysis of job 2.....   | 179 |
| Table A61 FAM analysis of job 5.....   | 179 |
| Table A62 FAM analysis of job 5.....   | 180 |
| Table A63 Detailed iterations on makespan.....   | 184 |
| Table A64 Results on performance indicators.....   | 185 |
| Table A65 Detailed results on makespan.....  | 186 |
| Table A66 Results on performance indicators.....   | 186 |

# Chapter 1 Introduction

A major change in the 21<sup>st</sup> century has been the impact of globalization. Even small and medium-size enterprises have manufacturing facilities in countries other than their home country. In some instances, this is a complex network of facilities. In others it's a single manufacturing subsidiary. Furthermore, non-core and core operations are being outsourced to countries with cheap labour. The implications of this are the diversity of environments within which the production planning and control system must operate has increased and will continue to do so. Organizations are also getting more complex these days. This complexity is due to several factors like complex production systems, product variety and uncertainty in business processes. The customer on the other hand still remains king in the competitive manufacturing environment. The capabilities of manufacturing plus expectations of customers has led to increased pressure for both speed and variety. Customers are demanding more tailoring in the products that they order and want them faster than ever. Part of this is derived from the expectation of shortened product life cycles, while part is derived from customers wanting more individualized treatment. These relationships must be incorporated in the production planning and control systems of the firms.

Efficient techniques for planning and replanning the entire supply chains to cope with the complexity and business dynamics are more and more apparent. Within the manufacturing and distribution supply chain, manufacturing scheduling and re-scheduling is one area, which will be vital to the success of the manufacturing organization. Especially, the day-to-day operations and the execution problems arising out of uncertain characteristics of the system have to be handled efficiently. Besides this today's ERP software products do not offer the possibility to model in a detailed way the underlying production system. There exists a gap between detailed scheduling and the execution of this schedule on the shop floor. There seldom exists a feedback mechanism to apprehend execution problems.

Simulation and optimization assisted planning and replanning of such systems becomes important as they offer the possibility to interactively analyze the system by testing the effect of execution exceptions and the resulting solutions without implementing them on the actual production system. This thesis addresses some aspects of combining simulation and optimization for scheduling and rescheduling of flexible manufacturing systems subject to several optimization constraints as well as meeting customer demands, whilst still addressing the questions of execution exceptions.

# Chapter 2 Problem areas addressed

## 2.1 Introduction

In the following sections, we address the specific problems and the sub-problems that are addressed in this thesis one by one. First problem areas and issues in predictive and then reactive scheduling system are described. The chapter concludes with optimization objectives of this study, assumptions and some important definitions. In the next chapter, we outline the state of the art in these areas.

## 2.2 Scheduling and re-scheduling of flexible production systems subject to execution exceptions

As one could imagine there are several configurations of production systems, each with its own characteristics. In this report, we seek to address scheduling<sup>a</sup> and re-scheduling<sup>b</sup> of one such flexible production system configuration – the flexible flow shop or more specifically the flow shop with identical, parallel machines at one or more stages. This problem is a combination of the classical flow shop-scheduling problem and the flow shop with multiple and identical machines at one or more stages problem.

### 2.2.1 Predictive scheduling of flexible production systems subject to known and unknown execution exceptions

Predictive scheduling consists of computing a production schedule prior to execution in the real world. The problem of predictive scheduling can be further divided into sub problems as follows:

#### 2.2.1.1 Approaches to predictive scheduling – combining simulation and optimization

There are several approaches of scheduling manufacturing systems. Discrete event simulation<sup>c</sup> and optimization<sup>d</sup> algorithms are increasingly used due to

---

<sup>a</sup> Scheduling, is defined as “specific overall assignment of resources on orders and the timed arrangement of the orders”, translated from Ten Hompel (2005).

<sup>b</sup> Re-scheduling, is defined as “scheduling anew at a time later than the earlier scheduling time”, reference [53].

<sup>c</sup> Discrete event simulation, is “the operation of an imitating system represented as a chronological sequence of events. Each event occurs at an instant in time and marks a change of state in the system”, Banks (1998).

<sup>d</sup> Optimization, is defined as “the study of problems in which one seeks to minimize or maximize a real function by systematically choosing the values of real or integer variables from within an allowed set. “, reference [53].

their relative simplicity in use and implementation. However, using purely simulation or purely optimization does not today address the complexity of existing manufacturing installations. For instance, pure simulation does capture all the detailed elements of the bigger system such as buffer<sup>e</sup> sizes, forklifts<sup>f</sup>, etc, but with the result that simulation itself does not optimize the system performance on factors like delivery time constraints<sup>g</sup>, make-span<sup>h</sup> of the schedule to be computed and other known events in a single run. On the other hand, optimization algorithms do not usually encompass all the details of the system, but can optimize and consider constraints on a system consisting of standard production system elements, and are relatively faster in their computations. An approach to manage the problem of considering details, optimization goals and constraints all together would be to combine optimization algorithms and simulation. How to combine simulation with optimization using *eM-Plant*<sup>i</sup> simulation software to consider some broader system elements (like buffers) and the above issues is partly addressed in this thesis.

#### **2.2.1.2 Fixed and flexible material flow routings with and without delivery time constraints**

Flexible manufacturing systems also have the ability to machine components and products with or without flexibility in routing due to machining constraints and at the same time with or without delivery time constraints. Clearly, this adds to the complexity of the system and problem which is addressed by the research in this thesis.

#### **2.2.1.3 Inclusion of known and unknown events and exceptions**

In the predictive planning phase itself, several known events can occur which influence how the production schedule is computed. Examples are tools, materials, maintenances, and resource availabilities. Additional complexity is added to the flexible manufacturing system configuration to consider such events. Further, it is important that the predictive schedule is problem free – that it is evaluated sufficiently in advance that no execution exceptions will occur due to the computed schedule.

#### **2.2.2 Rescheduling (or Reactive scheduling) of flexible production systems subject to known and unknown execution exceptions**

---

<sup>e</sup> Buffer, or temporary storage which can be placed between two production machines for containing parts, translated from Ten Hompel (2005).

<sup>f</sup> Forklift, is a material handling equipment which transports parts from one place to the other in the factory floor, translated from Ten Hompel (2005).

<sup>g</sup> Constraint, is defined as "a condition that a solution to an optimization problem must satisfy in order to be acceptable, reference [53]

<sup>h</sup> Make-span, or flow-time is defined in section 2.3.

<sup>i</sup> *eM-Plant* simulation software initially developed as SIMPLE++, Dangelmaier (1988).

Despite advances in preventive maintenance techniques, exceptions do occur rarely during the execution of a predictive schedule. Reactive scheduling refers to the adaptation of the schedule currently under execution on the shop floor due to exceptions such as part rejection in process, tool failure on the machine, machine breakdown, and buffers are empty or full (system blockage) or generic exceptions like energy not available at a work station. The problems addressed in this area include:

#### **2.2.2.1 Approach to rescheduling – combination of simulation and optimization**

The issue of what approach to use for rescheduling for a given exception and its frequency of occurring also becomes important. Given the requirement that a fast response is expected, using pure simulation can take excessive times to evaluate all alternative rescheduling rules or simulation runs but can consider the detail system elements. On the other hand, pure optimization can compute global rough solutions, and shorten the solution computation time, but cannot usually compute a detailed system. The problem addressed here is how to combine simulation and optimization for rescheduling a system which is subject to exceptions infrequently using *eM-Plant* simulation software.

#### **2.2.2.2 Approach to rescheduling – when to reschedule ?**

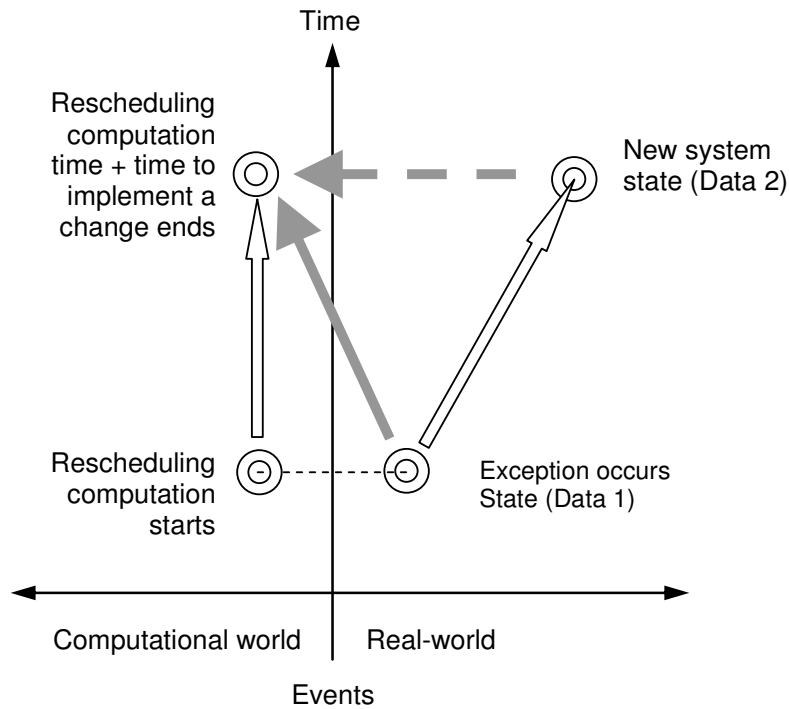
When exactly to reschedule in the physical production system is the problem addressed here. Especially which point in time to reschedule considering a graceful transition from a current system state of the real production process is considered. This has the following facets.

##### **2.2.2.2.1 Real-Time Control issues for rescheduling – Adaptation Synchrony**

Real-Time Control (RTC) pertains to a system or mode of operation in which computation is performed during the actual time that an external process occurs, in order that the computation results can be used to control, monitor, or respond in a timely manner to the external process. The external process can be characterized as each significant change from presupposed data. It can endanger an efficient or even a feasible execution of an existing schedule. This schedule is to be rescheduled using efficient techniques. Adaptation Synchrony define how the actual process continues during its modification (Bock (2005)). Production systems gain a certain momentum once they are operational. When changes to a schedule are made in response to exceptions, the system may never (or should not) be stopped entirely – parts of the system may proceed to another state (especially in the case of flexible and parallel machines), and a conflict may arise due to using



information from different states at different times during the change management process.



**Figure 2.1 Real-world state evolution and interaction with rescheduling**

The problem is shown in Figure 2.1. As seen in the figure when the rescheduling computation starts, the data at the time of the exception was considered for computation. When the rescheduling computations and implementations would be finished (assuming we can determine the times required for computations and the time needed to implement a change physically), the real system has evolved to a new state (corresponding to data 2). In this analysis, the problem of synchronizing the computations when dealing with such issues is undertaken.

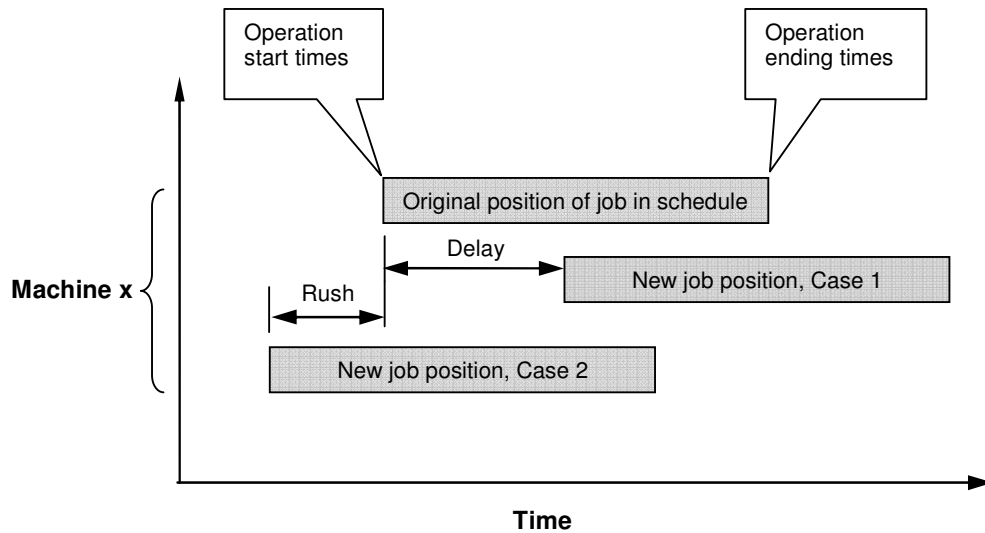
### 2.2.2.3 Rescheduling method

The rescheduling methods are further classified as:

#### 2.2.2.3.1 Deviation match-up with predictive schedule

In situations where manufacturing systems may operate (supplies of tools, materials, fixtures) on a just in time or just in sequence basis, or in environments where important resources such as material, tooling and fixturing are

delivered to the machine based on the initial schedule, it is essential that operation start times on each machine be adhered to as much as possible. This means deviations caused to the operation start times for each job on each machine due to the exceptions, should be brought back to their planned trajectory as much as possible. This method of rescheduling addresses this question. There are two factors associated with this namely reducing starting time deviations and reducing sequence deviations of the jobs as follows:



**Figure 2.2 Starting time deviation (devSt)**

### 1. Starting time deviation (devSt)

This is a very useful measure of the effectiveness of the re-scheduling algorithm. In this work, starting time deviation is measured by summing the absolute value of the differences in operation starting times between new and initial schedules as shown in Figure 2.2. This measure comprises two components:

*delay* = the sum of the absolute value of positive differences in starting times, and

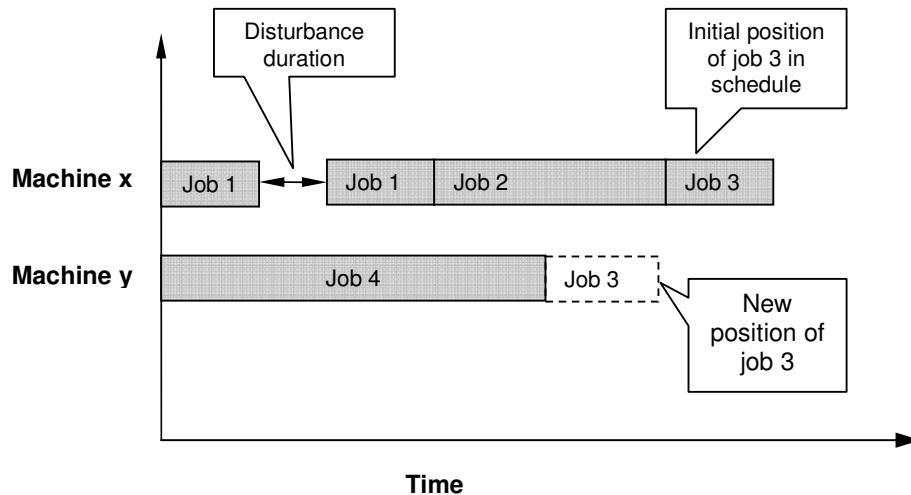
*rush* = the sum of the absolute value of negative differences in starting times.

*Starting time deviation* = *delay* + *rush*

### 2. Sequence deviation (devSq)

Sequence deviations are defined as the number of jobs re-scheduled (re-positioned) from their positions in the original schedule (Figure 2.3). This measure is critical if set-ups are prepared in advance based on the initial operation sequence on the machines.

For instance, jobs may wait on pallets in a sequence queue, and tooling and fixturing may be planned in advance according to the original sequence. Thus, a sequence change will incur efforts in resequencing the queue, reallocating the pallets, and replanning the tools.



**Figure 2.3 Sequence deviation (devSq)**

This can be formulated mathematically as follows: Let  $n$  be the total number of jobs on the candidate machine selected for rescheduling jobs. After re-scheduling, the number of jobs re located to other positions (from their original position in the initial schedule) from among these  $n$  jobs becomes  $n_r$ . If  $n_r = 0$ , then there is no sequence deviation. The percentage deviation from the original schedule can then be written as  $\left[ \frac{n_r}{n} \right] \times 100$ .

As can be imagined, both deviations can occur at the same time. If sequence deviation occurs (because of rescheduling), starting time deviation may or may not occur. If it does, it will have to be computed as per the description above. In addition to this, other measures of performance, like make-span and delivery times, may be affected due to the consideration of these factors.

### 2.2.2.3.2 Rescheduling as late as possible

Manufacturing systems may also work on the basis that rescheduling immediately after the exception may not be possible, as it may require immediate attention and resources. In such a case, an alternative method of rescheduling is to change as late as possible in the operating shift, without affecting the overall makespan too much. How to reschedule as late as possible is addressed here. It

uses the factor of sequence deviation to measure its effectiveness.

#### **2.2.2.4 Estimating the impact of the rescheduling step on future execution of predictive schedule**

Rescheduling might solve execution problems temporarily, but it may also affect future execution of a schedule. As an example, a rescheduling step at 1 pm, may cause problems at 2 pm, which when solved, may cause further problems at 4 pm. The question addressed in this is, how to assess the impact of a rescheduling step, and if there is an adverse impact, how to ascertain that we consider this impact in our computations of rescheduling for the initial exception.

### **2.3 Optimization objectives**

Both the predictive and reactive scheduling systems described in this report are designed to achieve the following key performance indicators (KPI). Schedules are generally evaluated by aggregate quantities that involve information about all jobs, resulting in one dimensional performance measures or KPI as we call them here. Measures of schedule performance are usually functions of the set of completion times in a schedule. For example, suppose that  $n$  jobs are to be scheduled. Aggregate KPI that are defined include the following:

#### **1. Make-span or Flow time ( $F_j$ ):**

The amount of time  $n$  jobs spend in the system, which is written as,  $F_j = C_{j=n} - r_{j=1}$ , where  $C_n$  is time the last job is complete and leaves the system while  $r_j$  is the time the first job arrived in the system.

#### **2. Equipment utilization (%): (Available machine hours\Scheduled machine hours) x100**

Since production schedules are obtained for a limited period (a time window), if the make-span is lowered, then within the same planning period, one could machine more jobs. Hence, a reduced make-span also results in better equipment utilization.

#### **3. Average in-process inventory levels ( $\bar{J}$ ):**

Besides flow time, one objective of scheduling is to maintain low inventory levels. Minimizing the inventory levels can be interpreted as minimizing the mean number of jobs in the system. In particular, the job sequence that minimizes the mean flow time will also minimize average in-process inventory (Baker (1974)).

Whether the vantage point is one of optimizing customer service or one of minimizing in-process inventory levels, the important problem is to find a sequence that minimizes the mean flow time  $\bar{F}$ .

### 5. Lateness ( $L_j$ ):

The amount of time by which the completion time of job  $j$  exceeds the time it is due:  $L_j = C_j - d_j$ , where  $C_j$  is the time at which the processing of job  $j$  is finished and  $d_j$  is the point in time at which the processing of job  $j$  is due to be completed. Delivering customer products according to the promised delivery dates is already a crucial factor in make to order<sup>a</sup> manufacturing environments. The aim of the processes developed in this thesis is to minimize the lateness as much as possible, during both the predictive and reactive scheduling steps.

## 2.4 Assumptions used in the study

The following assumptions are used in the study:

1. The system configuration considered is the flow shop with multiple, identical machines at each stages problem.
2. The number of jobs is known and fixed. No job, if in actual machining (processing) may be stopped before completion, unless an execution problem happens.
3. Jobs are classified as special jobs and standard jobs. Standard jobs are the ones which do not have constraints on delivery times and which machines they use in the production system. Special jobs are further classified as the ones which have only delivery times, the ones which have only routing constraints (use of specific machines on each stage) and the ones with both delivery times and routing constraints.
4. No two special jobs with similar or different routing can have the same delivery times.
5. In any configuration of the problem, standard and special jobs are roughly equally distributed to form the total number of jobs in the problem.
6. The arrival time, or release time, of all the jobs is known, is fixed or can be calculated.
7. The processing times of the jobs are known and constant.
8. Set-up times are independent of the job sequence and therefore are considered a part of processing time.
9. All jobs follow the same stage sequence, i.e. jobs flow from left to right.

---

<sup>a</sup> Make to order environments where products are manufactured only when the customer places a firm order.

10. Re-entrant flows within the manufacturing system are not considered.
11. Each job in the system is a unique entity, even though the job is composed of distinct operations, no two operations of the same job may be processed simultaneously.
12. Each stage has  $M_j \geq 1$  identical machines:  $j = 1, 2, \dots, m$ .
13. The flow shop consists of  $m \geq 2$  stages or levels.
14. All machines are available at the beginning of the planning window and the machines are continuously available, unless the machine maintenance times are pre-planned.
15. Schedule adaptation frequency is based on a execution problem oriented concept or event oriented concepts.
16. Execution problems do not occur every few minutes in the execution of the plan. When execution problems seldom happen, the plan is to be adapted according.
17. If there is an execution problem, the duration that the problem will last is known or can be estimated.
18. In-process inventory is allowed and is maintained in the buffer queues which have limited capacities.
19. For the purposes of rescheduling, each job is given equal priority. In other words, differences between jobs types are not made for rescheduling.

## **2.5 Real Time Control (RTC) terminology used in the report**

Some specific terminology has been developed to advance the techniques of RTC (Bock (2005)) used throughout the rest of this thesis, especially in the section of rescheduling system development. Adaptation handling defines:

1. Adaptation frequency – defining when an adaptation is realized.
2. Adaptation synchrony – defining how the actual real process continues during its modification.
3. Adaptation duration – defining the duration of the adaptation process. This policy will define the extent to which the adaptation is to be carried out.
4. Adaptation technique – defining how the adaptation is realized.

## **2.6 Structure of the report**

In this report, we discuss the state of the art in these areas namely, scheduling and re-scheduling for real-time control using approaches like simulation and optimization, followed by results of the state of the art, concepts and results for the developed approaches. The aim of the algorithms will be to work automatically and seek to optimize on key performance indicators. In chapter 3, we provide the

state of the art on these problem areas, namely scheduling, rescheduling and the use of simulation and optimization in these areas. In chapter 4, we state the result of the state of the art. In chapter 5, detailed solutions and methods are described to solve the problems addressed in this chapter, using small examples where possible, which are validated using the software application developed in chapter 7 and 8. Chapter 6 describes the synthesis of the total solutions. Chapter 7 describes the prototype software for the entire system including the system flow diagrams, how to start up the system and is explained with the help of an example. Chapter 8 assesses the optimization and simulation approaches quantitatively with the help of case studies, including validation of the approaches and algorithms used. Chapter 9 provides a discussion on the contributions of this thesis, conclusions from this study and presents further areas of research followed by acknowledgements and references.

## **Chapter 3 State of the art**

### **3.1 Introduction**

In this section, state of the art on the problem areas addressed in the earlier chapter are discussed. We first discuss the various approaches of predictive scheduling. The various methods include the simulation based and optimization based approaches used in industry. Then various approaches of rescheduling production systems subject to execution problems are discussed. These approaches also include optimization and simulation based approaches. Real-time control aspects and definitions are then presented followed by conclusions. In the next chapter the conclusions of the state of the art are discussed.

### **3.2 Predictive scheduling of flexible production systems**

#### **3.2.1 Various approaches of predictive scheduling**

Several methods of scheduling exist for different types of problem domains. If all numeric quantities are known in advance, then the problem is termed as deterministic scheduling problem. The ready time or the job release time makes a considerable difference in the problem structure. If all the jobs are assumed to be available at time zero, then the problem is termed as static problem. A non zero release time for a set of job defines as dynamic behaviour (Banks, 1998). Scheduling criteria may be classified into three categories such as shop performance based, due date based and cost based (Banks, 1998). The two most relevant approaches are discussed next in some details namely the analytical-optimal approaches and the simulation based approaches.

##### **3.2.1.1 Optimal-Analytical based approaches of scheduling the PMFS problem**

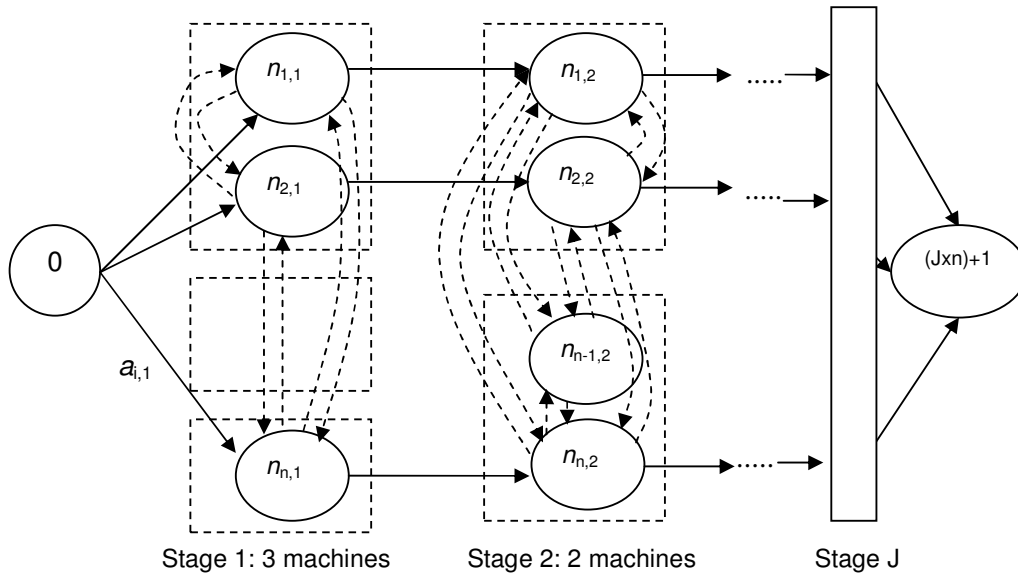
These approaches are also called as optimization based approaches. These approaches are designed to develop optimal schedules to minimize or maximize a scheduling criterion. The advantage of these approaches is that the scheduling criterion is explicitly considered during the development of a schedule. However, this requires that the quantifiable objectives be determined for the particular scheduling application. These approaches also require considerable amount of solution time to obtain an optimal solution if the number of alternative solutions is large. Therefore, some optimization-based approaches sacrifice from optimality in exchange for faster solutions. A fast solution may not consider all the possible alternatives explicitly but may choose from a subset of available alternatives and evaluate them before reaching a solution. These approaches are generally



considered as heuristics.

A Flow Shop (FS) is defined by Pinedo (2001) as a group of machine set up in series through which a number of jobs are processed and the operations are performed on each job in the same order, in that the jobs follow the same route. In the flexible flow shop, any stage in the flow-shop has more than one machine in parallel. The PMFS (Parallel Machine Flow Shop) problem is the task of sequencing these jobs through the flexible flow shop with respect to a certain objective. Technically, this problem is a combination of the flow shop scheduling and the parallel machine-sequencing problem.

The problem of the flow shop with multiple machines can be described as follows. There is a main incoming queue of parts (jobs), where each part has a different processing time, and each part can advance to any of the  $M_j$  machines at stage 1. Each job requires different processing times on any one machine on each of the stages. Theoretically all of the parts (except for special jobs) can be routed to any one of the machines at the next stage  $j$ . When the part has been processed through the last stage  $J$ , using one of the  $M_m$  machines, it is complete and can leave the system.



**Figure 3.1 A disjunctive graph for the PMFS problem**

Figure 3.1 shows a disjunctive graph in more details. There are  $n$  jobs to be processed and  $J$  is the total number of stages in the flow shop. Node  $n_{i,j}$  represents the processing of job  $i$  at stage  $j$ . The first node 0 represents the dummy node, indicating start of the operation. The last node  $(Jxn) + 1$  represents the completion of

processing of all the operations for the entire set of jobs. Each node is connected with the others using two sets of arcs, namely, directed arcs and disjunctive arcs. The directed arcs (bold lines with arrows) represent precedence constraints defined by the job routings and job sequences at various machines. Initially, every pair of jobs to be processed at a stage is connected by a pair of disjunctive arcs (dotted lines with arrows). As jobs are sequenced one by one, the disjunctive arcs between the jobs are replaced by directed arcs to represent sequence of jobs at different machines. All arcs from node 0 are directed arcs and have length equal to  $a_{i,1}$ , which is the arrival time of job  $i$  at stage 1. Each arc starting from node  $(i,j)$  is of length  $p_{i,j}$ , which represents the processing time for job  $i$  at stage  $j$ . All arcs ending at node  $(J \times n) + 1$  are of length  $q_{i,j}$ , which is the processing time for job  $i$  at the last stage i.e. stage  $j$ .

For such a system configuration, Gupta and Ruiz-torres (2000) have developed algorithms. The problem of scheduling  $n$ -jobs with release dates and due dates on parallel machines is shown to be NP-hard by Carlier (1987). Hence the problem of the parallel machine flow shop (PMFS) problem (combination of both problems – the Flow Shop (FS) scheduling and the parallel machine (PMS problems)) should be at least equally hard.

Relatively limited amount of work has been published in the area of the parallel machine flow-shop scheduling problem. Brah and Hunsucker (1991) have developed a branch and bound algorithm for the PMFS problem. Brockman and Dangelmaier (1997) also developed a branch and bound algorithm for the PMFS problem using parallel processors (computers) to solve the problem. These algorithms were noted to have worked consistently with a fair amount of computational speed for small to medium sized problems. For instance, Brockman and Dangelmaier report computation times of 15 seconds using 1024 parallel computers for 11 jobs, having 3 stages and 3 machines on each stage, and 15 hours using 16 parallel computers. For large sized problems, further improvements were suggested. One of the relatively recent advances in this field of *job shop scheduling* comes from the shifting bottleneck procedure by Adams et al. (1988). This procedure takes advantage of a very efficient algorithm developed by Carlier (1982) for scheduling jobs with heads and tails on a single machine. The shifting bottleneck procedure identifies bottleneck stages (critical stages) in the *job-shop* using the Carlier algorithm and sequences the stages one by one. The procedure has been found to produce excellent results in very little computational time. However, the success of this procedure depends on exploiting the effectiveness of the Carlier's algorithm. More recently, Cheng et al. (2001) developed a heuristic using a combination of a property of the PMFS problem and the shifting bottleneck procedure. Phadnis et al (2003) also developed such an algorithm for the PMFS problem using the shifting bottleneck procedure, and have compared their results

with other algorithms, namely that of Cheng et al. and others. They developed such an algorithm in which they used the very efficient shifting bottleneck (of Adams et al, 1988, applied for the *job shop scheduling* problem) procedure and applied to the Parallel Machine Flow Shop problem with the possibility of producing equally good results. As this procedure is used partly in the predictive part of the thesis, it is described here briefly.

Step 1 Set  $I_0 = \Phi$ . Get processing times ( $p_{i,j}$ ) at each stage and job ready times at stage-1 ( $a_{i,1}$ ) for all jobs to be scheduled ( $i$ ).

Step 2 Calculate tails for each job at each stage, using the formula  $q_{i,j} = \sum_{j=m+1}^J p_{i,j}$ .

Tails are computed to find out the amount of work left at a certain stage.

Step 3 Calculate the earliest start times ( $s_{i,1}$ ) at stage – 1 for all jobs yet to be scheduled. The earliest start times for a job will depend on when one of the machines at stage 1 will become free and when the job arrived. This can be written as:

$$s_{i,1} = \max \{ (a_{i,1}), \min \{ a_{1,k}^m \mid k=1,2,\dots,m_1 \} \} \quad \forall i \rightarrow I'$$

In effect, a selection of the maximum of the job ready times and minimum of the machine ready times is done.

Step 4 Calculate the earliest start times ( $s_{i,j}$ ) at each downstream stage – 1 for all jobs yet to be scheduled using the formula:

$$s_{i,j} = \max \{ (s_{i,j-1} + p_{i,j-1}), \min \{ a_{j,k}^m \mid k=1,2,\dots,m_j \} \} \quad \forall (i \rightarrow I', \text{ and } j=2,3,\dots,J).$$

Step 5 Calculate the bottleneck or critical stage among all the stages. A critical stage is one, which calls for extra care to be taken to schedule a job. In other words, we find critical stage, and schedule a job such that the resulting make-span is as less as possible. In order to calculate the critical stage and to evaluate the result of the heuristic, the optimal makespan needs to be calculated (resulting from an optimal schedule). However, because the optimal schedule is unknown, the minimum possible make-span needs to be calculated. This minimum value is referred as lower bound of the problem. There are several techniques to compute the lower bounds for the single-machine sequencing problem. In this case, the lower bound on the makespan for the single stage parallel machine problem is used to find the critical stage, instead of using optimal makespan for the one-machine sequencing problem (as is done for the job shop problem). In this heuristic,

the lower bounds on makespan are calculated for all stages based on the jobs not sequenced yet. Then the stage that has the highest value of lower bound is chosen as the critical stage. Figure 3.2 shows the concept.

The lower bounds are similar to the ones used by Carlier (1987) and Gupta and Ruiz-Torres (2000). Two lower bounds are computed and the bigger of the two is used to select the critical stage.

Lower Bound 1<sub>j</sub> = max (s<sub>i,j</sub> + p<sub>i,j</sub> + q<sub>i,j</sub>) where, i ∈ I - I<sub>0</sub>, for each job i = 1 to n.

Lower Bound 2<sub>j</sub> =  $\frac{1}{m_j} ((m_j \text{ earliest job start times}) + \sum_{i=1}^n p_{i,j} + (m_j \text{ shortest tails}))$

Lower Bound<sub>j</sub> = max {Lower Bound 1, Lower Bound 2}

Lower bound 1 is computed with the assumption that the jobs will be processed one after the other on the stage (by assuming that there are more than n machines in the stage). This way, the highest value of the lower bound for each job, will be the lower bound for the stage. On the other hand, lower bound 2 is computed in such a way that there are less than n machines or that the other machines are so busy that all jobs have to be processed on one machine. The actual lower bound for the stage is the maximum of the two bounds. This procedure is carried out for each stage, and the critical stage is the one with the highest value of the lower bound. Hence,

Lower bound = max (Lower Bound<sub>j</sub> | j = 1, 2, ..., J)

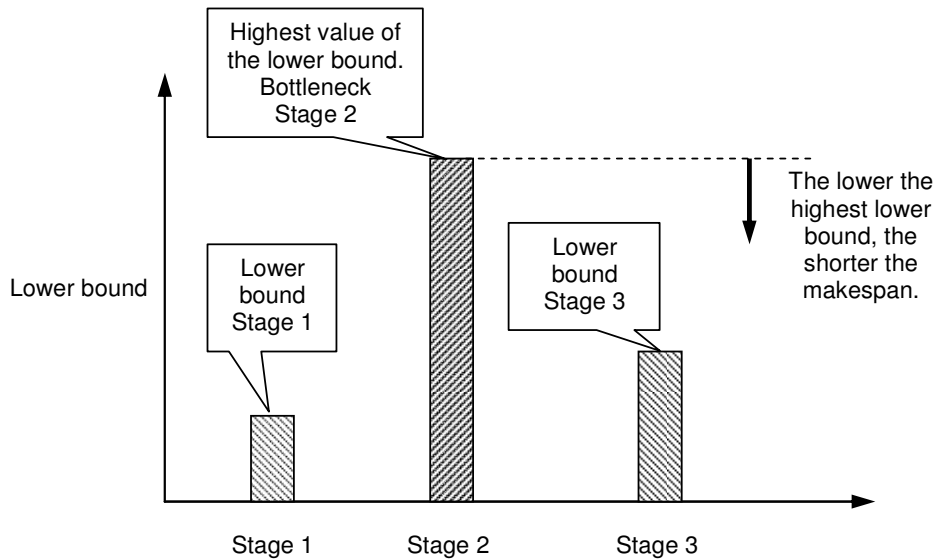


Figure 3.2 Highest lower bound stage as critical stage

Step 6 Find the job to be scheduled as follows:

- a. Set  $t = \min \{ a_{BN,k}^m \mid k=1,2,\dots,m_{BN} \}$
- b. Let  $I''$  be the set of jobs not yet scheduled and available at time  $t$ .  
 $I'' \subseteq I'$ , where  $s_{i,BN} \leq t$
- c. If  $I_o = \Phi$ , then set  $t = \min \{ s_{i,BN} \}$ , where  $I \rightarrow I'$ . Go to step 6a.

In this step, the clock is set to the earliest possible start time for a job at a critical stage. Then the heuristic, goes back to step 6a, since there could be more than one job. In the next step from all these jobs, an appropriate job is selected for scheduling.

- d. From among the available select job  $i'$  with the longest tail:  
 $i' \rightarrow I'$  and  $q_{i',BN} \geq q_{i,BN} \quad \forall i \rightarrow I''$ .

In case of a tie, select the job with the longest processing time at the critical stage:

$$p_{i',BN} \geq p_{i,BN} \quad \forall i \rightarrow I''$$

Here, the job with the longest tail or the longest processing time (LPT) is selected to prioritize the job with maximum work remaining. LPT rule is proved to provide good results (Baker (1974)).

Step 7 Schedule this job  $i'$  depending on the result of the previous step, at **all** the stages:

- a. At each stage ( $j$ ), find the earliest available machine  $m'$ .

$$a_{j,m'}^m = a_{j,m'}^m + p_{j,m'} \quad \forall (k=\{1,2,\dots,m_j\} \text{ and } j=\{1,2,\dots,J\})$$

Schedule job  $i'$  at machine  $m'$ .

- b. Update the ready time for machine  $m'$  as

$$a_{j,m'}^m = a_{j,m'}^m + p_{i,j} \quad .$$

At this stage, the selected job, is scheduled "horizontally" through all the stages, on the earliest available machine at each stage respectively. After scheduling this job, the ready times for the scheduled machines are updated. So the ready times for a particular machine on all stages will be used in the second iteration to schedule the second job.

Step 8 Update the number of scheduled jobs as  $I_o = I_o + \{i'\}$ .

Step 9 If all jobs are scheduled, go to step 2, else, go to step 10.

Step 10 End.

This algorithm when compared to other algorithms for the PMFS problem generated better results for makespan as laid out in Table 3.1.

**Table 3.1: Comparison of heuristic with other approaches**

| Dataset | Makespan        |              |           | % Deviation from LB |              |           |
|---------|-----------------|--------------|-----------|---------------------|--------------|-----------|
|         | Wittrock (1988) | Cheng (2001) | Algorithm | Wittrock (1988)     | Cheng (2001) | Algorithm |
| 1       | 784             | 764          | 760       | 7.54 %              | 4.80 %       | 4.25 %    |
| 2       | 789             | 773          | 770       | 6.33 %              | 4.18 %       | 3.77 %    |
| 3       | 785             | 764          | 770       | 5.94 %              | 3.10 %       | 3.91 %    |
| 4       | 796             | 789          | 785       | 7.71 %              | 6.77 %       | 6.22 %    |
| 5       | 964             | 963          | 961       | 0.31 %              | 0.21 %       | 0 %       |
| 6       | 686             | 669          | 667       | 5.70 %              | 3.08 %       | 2.77 %    |

The approach described above (and approaches of other researchers based on Branch and Bound, etc) considers only processing times for jobs, the number of machines and stages for computing a schedule to optimize on make-span and unlimited buffer capacities between two stages. Consider a system when additional system elements such as buffers with limited capacities, and delivery time constraints for some jobs, additional routing constraints for some jobs which are special, along with known events about resources, tool and materials availabilities, and solving problems which may occur when executing a schedule in the real world are added to the problem description. To solve the problem of computing a minimum make-span, and meeting the above goals, then becomes too complex for such algorithms to consider all by themselves.

### 3.2.1.2 Simulation based approaches for predictive scheduling

Today there exists several simulation based planning and scheduling software on the market place. In the following discussion, characteristics of each in relation to the problems addressed in thesis are described.

#### 3.2.1.2.1 AutoSched: Simulation based scheduler

*Brooks Automation inc.* offers *AutoSched* software as one of their product suites. Using *AutoSched*, planners can experiment and test ideas for improving fab, assembly, or test facility, such as new scheduling rules, product mixes, start rates, etc., without the expense and impact of trying ideas on the real factory. Once they have proven a new dispatching policy using simulation, they can implement the new policy in the real facility using a Real Time Dispatcher. *AutoSched* is used to make those difficult-to-answer planning decisions, such as:

1. When do I start a lot?

2. Is the lot behind schedule? When will the lot be completed?

3. How do I match sales orders to WIP inventories?

AutoSched uses the same dispatch rules as the Real Time Dispatcher, so the factory executes the same plan that was committed to customers. *AutoSched* is claimed to provide a flexible environment in which to implement any rule or policy needed, from the very simple to the very complex. Any policy that is based on data available from the MES and associated information systems can be implemented in the product. The dispatching rule engine is claimed to be extremely fast and can make even complicated "Which lot?" decisions in a fraction of a second. The *AutoSched* AP product is an object-oriented modeling tool, uses a Windows-based Excel spreadsheet interface and is integrated with the Real Time Dispatcher.

A brief comment is made here regarding the capabilities of *AutoSched* software. There doesn't seem to be a connection between optimization, conditions considering broader system elements, rules, and the result of applying such rules *during* the simulation run. As an example, there could be a condition like "if job is type  $x$  on machine  $z$ , then the tool  $t$  needs to be changed, which needs additional time of  $y$ ", which could be checked easily during the simulation run, but harder with an optimization algorithm especially if the optimization algorithm already does a lot of complex calculations. It seems that several simulation runs are required to be made with *AutoSched* to check the variation of one or more classical scheduling rules (classical scheduling rules such as FCFS, FIFO, etc) and constraints. Besides this, there is no mention how to enhance the performance of a system using rules.

### **3.2.1.2.2 SIMUL8-Planner: Simulation based planning and scheduling**

Recent literature also mentions about industrial attempts to develop simulation based production planning and scheduling software. Tremble et al. (2003) report the development of *SIMUL8-Planner* for simulation based planning and scheduling. There are different ways they apply *SIMUL8-Planner*. A typical approach is to use a pre-simulation planning routine followed by simulation based schedule generation. The following sections provide a brief summary of this method.

#### *1. Pre-simulation planning*

Traditional ERP systems plan work across a production facility using an aggregate, or timed-view of machine and resource capacity. Customer's orders or products are offset against available finished goods inventory to produce time phased production requirements. These requirements are generated from the bill of materials, process sheets, and work-in-process inventory data. Where production capacities are exceeded, work orders are assigned to earlier periods. This master planning approach does not ensure that orders will be completed on time when

executed on the shop floor. SIMUL8-Planner uses a similar approach, prior to running a simulation. The simulation considers the physical constraints within the system along with the softer management rules or preferences with regards to order priorities and production targets.

## 2. *The planning heuristic*

There are several different heuristics that can be applied during the planning portion of *SIMUL8-Planner*. However, the typical method of scheduling jobs or production using the software is to first create a plan using intelligent heuristics based on user defined objectives and dispatching rules. Creation of the plan is centered around identification of the critical process or bottleneck based on overall capacity, total planned usage, user-defined weightings for changeovers, due date conformance, and other decision rules that will impact upon the dispatch of jobs. A backward plan explosion through the bill-of materials is then used to allocate jobs within the plan. This process can be iteratively continued until all machines have been backward planned.

## 3. *Simulation schedule generation*

A key advantage of *SIMUL8-Planner* lies in its ability to quickly execute a production plan generating a feasible and efficient schedule. A production plan, on its own as described above, lacks the level of detail to ensure that it is both feasible and efficient. Executing a plan through a simulation of the plant processes results in a production schedule that includes job start and finish times by machine. It also provides service levels and manufacturing KPI. The production schedule is presented through interactive Gantt charts as described earlier with the facility for the end-user to manually change or re-sequence allocated work at the machine level. After inspection of the resultant schedule and plant KPI's, the user may elect to adjust the scheduling rules that control the simulation's execution of the plan. This may include order due-date conformance, the minimization of change-over, the level of work-in-process build-up, etc. These rules can be weighted to produce an overall schedule preference that is applied within the simulation of the plan.

To conclude briefly here, products like *SIMUL8-Planner* consider constraint and "softer management rules" during the scheduling process using an intelligent heuristic. They then simulate the resultant plan and based on the result adjust the rules to again run the simulation. If the production system is more complex, then the adjustment of rules to produce a better result than the previous simulation runs would be quite difficult because of the complex interrelationships between broader system elements. Secondly, rules and constraints are embedded in the scheduling heuristic. If a production system is more complex, this can make design and



evaluation (solving) of a heuristic quite hard due to the dynamic interrelationships that exist between broader production system elements.

#### **3.2.1.2.3 ISSOP simulation based scheduling system**

Krauth (2005), discuss industrial projects involving optimization and simulation in a combined way. They mention about a project of simulation based scheduling of an automotive painting process. They mention that today nearly every car in automotive production is different because of specific customer requirements. In the production step of painting, more than 100 different colors are possible. Combined with different car shapes like standard or station wagon, engine varieties and other options, the problem of finding the optimal product sequence is extremely difficult.

They mention that in contrast to the assembly process, where tools and process steps are very similar and uncritical in sequence, the painting process is critical for the following reasons:

1. Each color change requires a cleaning operation consuming considerable extra time and cleaning supplies.
2. The cleaning process is easier and faster if colors are changing all the time from a bright to a darker color.
3. The buffers in front of and behind the painting station are limited in space.

Therefore an optimized sequence of painting operations must be calculated, which meets the buffer restrictions and minimizes the consumption of resources and time due to color changes. An improved batch sequence can lead to major savings in costs and time. A graphical process model was developed. It was very helpful in internal discussions among the manufacturer's staff, and in discussions with the external simulation experts. The validated model was integrated with other IT systems so that the model always reflects the actual state of the real process. To generate and optimize schedules which satisfy all constraints, an intelligent optimization toolkit ISSOP was then coupled with the simulation model as shown in Figure 3.3.

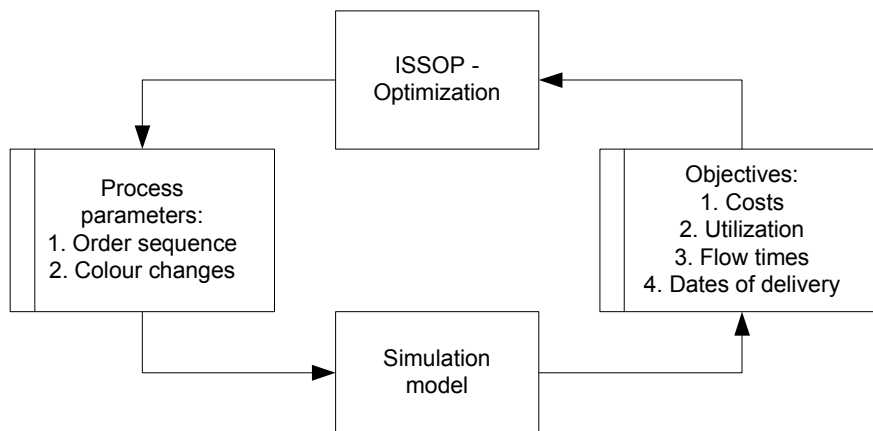
The optimizer ISSOP generates production sequences, the simulation tool uses them as input, it checks if all constraints are met and calculates costs and throughput times. ISSOP then compares the results and generates better sequence; the simulation model evaluates them again, and so on. After some iterations (the exact number is not mentioned), which take between 5 and 25 minutes (again, the size of the system and model is not mentioned), they mention that an improved sequence can be found which is then used for controlling the real process.

This approach, according to the developers of this system, increases the reliability of planning, reduced material consumption, throughput times and work in

progress. Additional advantages of the simulation model are the possibility of testing the schedule against execution problems such as technical problems or delivery delays. Although the production sequences were already calculated by software programs before, the results achieved by the new simulation and optimization approach were significantly better as recorded below:

1. 8 % less changes of color in the painting station.
2. Less color changes and more optimized batch sequence with 12 % higher output of the painting station.
3. Less losses of coating material
4. Less cleaning material needed and less critical situations because of less manual operations.

To conclude this discussion, this system is primarily “driven” by an optimization system, and simulation is simply used to check the constraints. Fine tuning is always done by the optimization iterations. For complex and bigger system configurations, this would result in several optimization and simulation runs. It is unclear how the criteria are determined which form an input to the next optimization algorithm run. On the other hand, if the production sequences are previously calculated by the optimization software, then these sequences have to be already optimal if the overall good solution is to be ensured. Further, how simulation is used to fine tune the system especially if the simulation encounters problems like bottlenecks and constraints during run time is also unclear.



**Figure 3.3 Interaction of simulation and optimization tool for car painting (Krauth, 2005)**

#### **3.2.1.2.4 Preactor International case of simulation based scheduling**

Krauth (2005) further mention about a practical use of simulation and optimization in a combined way at another practical production set-up. They

mention about the work of *Preactor International* for a medium sized company which supplies small pressed aluminum parts to a range of other consumer focused businesses. In this work, *Preactor* used its scheduling tool to generate an “optimal schedule” followed by using simulation to model a certain process of a plant in a detailed way to result in quantitative information about performance indicators. They further mention that they arrive at a final production schedule by simulating several scenarios offline, resulting in savings in production rates, and increased revenues.

To conclude this discussion, this approach of simulating several scenarios does not guarantee the trouble free execution of the schedules. Fine tuning or improving the plan obtained from the optimization algorithm is not considered at all.

### **3.3 Reactive scheduling of flexible production systems**

Over the last two decades, a significant volume of research on the issues of scheduling with executional exceptions has begun to emerge. A review of this research is presented here, starting with approaches of reactive scheduling, and associated nuances, and situation when they are used or are appropriate followed by describing when to reschedule, and real-time control issues.

#### **3.3.1 Various approaches to reactive scheduling**

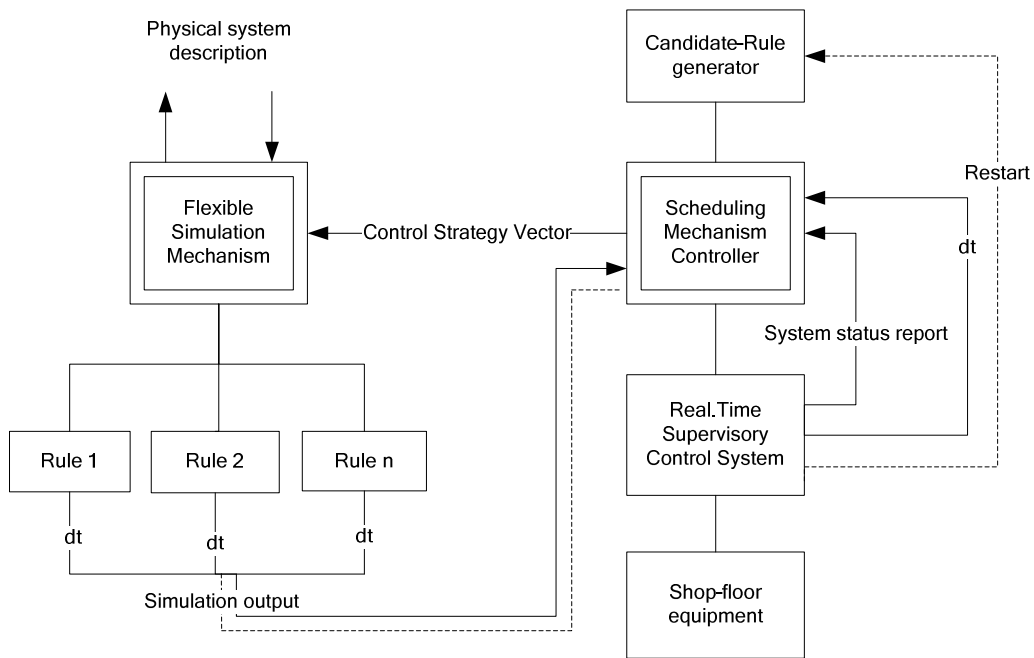
In this section different approaches are discussed using simulation and other approaches.

##### **3.3.1.1 Simulation based completely reactive approaches**

This category of modeling approaches does not take any of the cause, and impact into consideration per se. These completely reactive approaches are characterized by least commitment strategies such as real-time dispatching that create partial schedules based on local information. Dispatching (Bhaskaran and Pinedo, 1991) examines the jobs currently available at the machine in question, and sometimes in its immediate vicinity. The next job to be processed is selected from among these by sorting and filtering them according to the pre-defined criteria, and selecting the job at the head of the resulting list. This approach has many practical advantages. Its computational burden is in general extremely low, and the rules are usually intuitive and easy to explain to users. A number of the more sophisticated dispatching procedures can invoke complex rules that allow them to consider the state of the system, at several different machines, and to take conditional actions based on this state. This type of policy has been extensively implemented in the semiconductor industry (Haldun, A et al. 2005).

An extension of the dispatching approach is to allow the system to select dispatching rules (a dispatching rule is a function  $f$  which assigns to each waiting

job  $i$  a scalar value, the minimum of which, among waiting jobs in the system (i.e. in a machine queue, input queue or on a machine), determines the jobs to be selected over all others for sequencing) dynamically as the state of the shop changes. Early work in this area is that of Wu and Wysk (1989) (Figure 3.4), who examine the problem of dispatching rule selection in the flexible manufacturing system (FMS) environment. They divide the time horizon into shorter intervals  $dt$ . At the beginning of each interval a variety of dispatching rules are simulated (first level decisions shown by thick lines, and second level decisions shown by dotted lines), and the rule that yields the best performance is selected and implemented for the next time period. They also mention that decisions are made locally, and may not contribute to the global system performance. This means that if the system is highly dynamic or otherwise, they do not make sure global performance is considered.



**Figure 3.4 General scheme of simulation based reactive scheduling (Wu and Wysk, 1989)**

They also do not compute the impact of the exception on the global system performance. This is important to determine whether a rescheduling action is required. A number of other authors study how to estimate the impact of an exception at a particular point in the execution of a predefined schedule. Many of these papers use algorithms (not simulation) that can be inferred from the well-

known disjunctive graph representation. A number of researchers (notably Abumaizar and Svestka (1997) have used the principles of the graph to compute the effect on operation start and end times using longest path calculations in the graph, by updating the duration of the operation during whose processing the exception occurs. They perform any but simple calculations based on exceptions. For a flexible production system with complex job routings, it would indeed be very hard to update the operation start and end times accurately enough to further use the end result of the computation effectively.

Wu and Wysk (1989) also mention in their paper on how time consuming their experiments are. They mention that if several combinations of rule simulations are made, it would require 18,720 simulation cycles. Most importantly, they mention that their work considers only one branch of the FMS configuration to avoid the overwhelming computation time to give some preliminary results for their conjectures.

A number of authors have extended the approach of Wu and Wysk in various ways, e.g., Kim And Kim (1994) and Jeong and Kim (1998). Harmonosky et al. (1997) present their work in the areas of real-time selective re-routing and rescheduling algorithms based on simulation. They iteratively use simulation as a tool to find out the best policy from a set of alternative policies in real-time. Using simulation as a tool for real-time scheduling presents several benefits and drawbacks. The most important benefit is that simulation can provide accurate information about a certain policy. Secondly, simulation proves to be effective when the system behavior cannot be easily captured by analytic methods. The ability to provide accurate information about a certain policy becomes a drawback when it comes to modeling a complex manufacturing system. As a manufacturing system gets bigger and as the number of “control points” (location where a decision to handle a part has to be taken) increase with each point providing several alternatives (due also to part variety), the more difficult it is to employ simulation, especially in real-time to obtain information about the best alternative policy. An extension to this work has been in the areas of simulation and machine learning. Piramuthu et al. (1991), who first use a simulation model of the manufacturing system under study to develop a characterization of how different dispatching rules perform in the system under different operating conditions. They then apply an learning algorithm to this data to develop a decision tree that selects a dispatching rule whenever a significant change in system state is identified.

In a yet different way, Manivannan and Banks (1991) present a simulation and knowledge-based Real-Time Control system for flexible production systems.

Manivannan and Banks (1991), mention that a Real-Time Control (RTC) system must be capable of the following:

1. Reacting to the problem instantaneously.

2. Evaluating several alternatives policies.
3. Providing optimum or near optimal solutions.
4. Learning from previous problems.
5. Providing faster and more accurate solutions.

They present a sophisticated control architecture of a knowledge based RTC system using simulation. The aim of the framework is to provide an integrated environment for the controller to evaluate various control policies using simulation. Data for simulation is collected from the manufacturing cell. The main issues they address and develop in their system is the synchronization of the events between the simulation system and the real-system as a means to provide instantaneous feedback\control instructions. A temporal knowledge base has been designed to synchronize the events and their times of occurrence in both the manufacturing cell and the simulation model. Also, a dynamic knowledge base has been implemented to store simulation results. They claim that this feature provides a faster response to a control problem by reducing the number of resimulations conducted for evaluating various alternative policies in real time. The focus is however purely on how to provide a feedback in the fastest possible way and not considering the fact that real world will nevertheless evolve and that the controlling mechanism needs to devise a method which effectively recognizes this evolution.

### **3.3.1.2 Predictive – reactive scheduling**

In predictive-reactive scheduling, scheduling is presented as a two-step process. First, a predictive schedule representing the desired behavior of the shop floor over the time horizon considered is generated. This schedule is then modified during the execution in response to execution exceptions. The schedule that is actually executed on the shop floor after these modifications is called realized schedule. The two main questions are when to initiate a rescheduling action and assessing the impact of a given exception on existing schedule. Hence, our discussion in the section will begin by examining the issue of when to initiate a rescheduling activity.

#### **3.3.1.2.1. When to reschedule**

Regarding this question, when to reschedule, the basic question that needs to be answered is when a disruption or an event has sufficient potential impact that a new schedule must be generated or some more localized remedial action taken. Church and Uzsoy (1992) provide a rough taxonomy of existing approaches beginning with two extremes. Continuous rescheduling approaches take rescheduling action each time an event that is recognized by the system, such as the arrival of a new job, occurs. Periodic rescheduling, on the other hand, defines a basic interval  $T$  between rescheduling actions during which rescheduling

actions are not permitted. Rescheduling actions are taken at times  $kT$ , where  $k$  is an integer. These points in time where rescheduling may be performed are referred to as rescheduling points. Any events occurring between rescheduling points are ignored until the following rescheduling point. Finally, they define event-driven rescheduling, in which a rescheduling action can be initiated upon the recognition of an event with potential to cause significant disruption to the system. Both continuous and periodic rescheduling can be viewed as special cases of event-driven rescheduling.

Clearly, continuous rescheduling runs the risk of initiating rescheduling activity in the face of events that do not cause significant disruption, expending computational resources and potentially causing unnecessary changes in the schedule with associated ill effects on the shop floor. The obvious drawback of periodic rescheduling is that it ignores events occurring between rescheduling points, which in an extreme case may render the current schedule impossible to execute, and in less serious situations runs the risk of yielding poor schedules. Hence they mention that a combination of the periodic and event driven approaches appears attractive, in which a periodic rescheduling approach is implemented, but rescheduling activity can be invoked between rescheduling points if a disruption that is deemed sufficiently serious is observed. This latter approach is more commonly observed where schedules are often developed for some base horizon, such as a day or a shift, but are modified as needed during that period.

A number of authors have adopted the periodic and event-driven view of rescheduling and analyzed different approaches in this area. Church and Uzsoy (1992) consider the problem of minimizing maximum lateness on single-stage production systems involving single and parallel machines, where the only source of uncertainty is random job arrivals. Their results indicate that schedule quality initially improves quite rapidly with more frequent rescheduling, but after a certain point yields almost no further gains. This is intuitive, since once the frequency of rescheduling activity exceeds the frequency of disruptions to the system the rescheduling activity is merely causing nervousness without improving the schedule quality. Another way of putting this is that a periodic response may well be sufficient to deal with the disruptions faced by the system, and that rescheduling with every system state change may be counter productive. These results have been supported by a number of subsequent researchers for a variety of shop floor environments, e.g. Sabuncuoglu and Karabuk (1998) for flexible manufacturing system with uncertain job processing times and machine breakdowns; Shafaei and Brunn (1999a) for open shops and others.

Most researchers have focussed on when to reschedule by using either periodic rescheduling, continuous rescheduling, event oriented rescheduling, or a combination. None of them consider graceful transition of execution issues like

the exact time when to reschedule in the real system considering its continuous evolution to another state.

### **3.3.1.3 Predictive-reactive scheduling versus completely reactive approaches**

A number of authors have examined the question of when a periodic or event-driven rescheduling policy based on a global view of the scheduling problem can perform better than a completely reactive dispatching approach. Yamamoto and Nof (1985) compare the effects of a fixed optimization based schedule, an event rescheduling approach and dispatching rules in an FMS environment. They find that in the systems under study, a fixed optimization-based schedule obtained from a branch and bound algorithm outperforms myopic dispatching rules in the face of machine failures, and is in turn outperformed by the event-driven rescheduling approach. Hutchison and Khumawala (1991) examine this question in a FMS environment where the only uncertainty is due to job arrivals at the start of the planning periods. They find that a periodic rescheduling policy based on their optimization formulation outperforms dispatching, especially when there is routing flexibility. Wan (1995) shows that when processing times are variable, a global scheduling algorithm may yield poorer solution than a dispatching policy. An important paper in this area is that of Lawrence and Sewell (1997), who compare the performance of global scheduling heuristic based on shifting bottleneck algorithm of Adams et al. (1988) with myopic, completely reactive dispatching rules in the presence of varying job processing times. They demonstrate that as processing time variability increases, the difference in performance between the global method and the dispatching rules becomes less significant. They conclude that in systems with high execution exception frequencies, completely reactive algorithms can be used with relative confidence, and question the benefits of global scheduling procedures in general.

Matsuura et al. (1993) provide an extensive study of a slightly different rescheduling policy. In their approach, called switching, a predictive schedule is developed on a periodic basis. However, if the realized schedule is deemed to have deviated sufficiently from the predictive one, the system switches to using a dispatching rule for the remaining period. This approach is contrasted with using the predictive schedule throughout the period (by right-shifting jobs when delays occur) and dispatching approaches. They focus three different types of disruptions: rush order arrival, specification changes (which cause new operations to be added to a job, or existing operations to be deleted), and machine failures. Their results are quite insightful: they show that when the frequency of execution problems is low, the predictive \ reactive approaches outperform the dispatching. Once the level of execution problems reaches a certain level, however, the dispatching begins to



perform better than the predictive \ reactive approaches.

The answer to this debate lies in the results of Matsuura et al. (1993), Lawrence, and Sewell (1997), and is hinted at in the results of several other papers. In an environment with low frequency of exceptions, predictive \ reactive methods based on global information and optimization techniques is highly likely to yield better schedules than completely reactive dispatching procedures. However, once the variability in the system exceeds a certain level, which appears to be system dependant, global information on which the predictive \ reactive approaches are based becomes invalid, causing them to generate poor schedules due to solving the wrong problem: the problem data they use does not correspond to the problem encountered on the shop floor.

Having agreed with Lawrence and Sewell (1997), it may even be sensible to pursue further work on the predictive \ reactive scheduling methods. This is because, when a manufacturing system is subject to high frequencies of exceptions it might be advisable that managements time and resources would be better spent on working to reduce it, rather than developing sophisticated scheduling logic (such as those using completely reactive and other robust approaches). In addition, there are many manufacturing systems in which the difference in performance that can be obtained from a sophisticated scheduling procedure over a dispatching rule is simple not worth the amount of trouble that would be required to implement the global system. On the other hand, in capital-intensive environments, which requires hundreds of unit processes and complex machinery and product routings, improvements of even a few percentage points in performance measures such as the average lead-time may be worth millions.

It is interesting to note that a number of researchers have attempted to use global schedules as a complement to dispatching rather than to replace them. In the approach described in Haldun et al. (2005), in which a global schedule generated on a periodic basis is used as a priority index in a dispatching rule that outperforms myopic rules that do not use global information under a wide range of operating conditions. In these groupings of research results, the scope of execution problems considered are limited and they are mostly random. The cause of exceptions is often machine availability (breakdown and repair) or some stochastic aspect of processing time that makes the start and finish times variable.

### **3.4 Conclusions**

From the state of the art, the following inferences can be made about the predictive scheduling system for the problem areas discussed:

1. The parallel machine flow shop problem is harder to be solved by an optimization algorithm *alone* when,

- a. Considering additional broader system details such as buffers with limited capacities, and at the same time,
  - b. Considering demands on routing of special or standard jobs through the system, both with or without constraints on delivery times, and at the same time,
  - c. Considering resource availabilities like tools, materials, and machine availability times set by maintenance plans in the planning horizon, and at the same time,
  - d. A problem free schedule is to be generated.
2. Simulation based planning and scheduling systems mostly work iteratively, using optimization algorithms first to generate a plan, and then using simulation to execute the plan to generate a schedule. The optimization algorithm mainly serve to drive the next iteration, while simulation simply simulates to compute starting and end times of the schedule. Fine tuning the schedule is not considered considering broader production system elements.

From the state of the art the following inferences can be made about the reactive scheduling system for the problem areas addressed:

1. Optimization algorithms are harder to be used alone to compute a rescheduling solution to a PMFS problem, especially to accurately compute the result of the exception.
2. Simulation based methods when used alone for rescheduling are not completely efficient as regards to their longer computation times and inability to compute rescheduling solutions which are optimal.
3. Rescheduling systems in literature do not consider Adaptation Synchrony part of RTC in their computations.
4. Rescheduling systems do not consider matching up deviations to the planned trajectories.
5. Rescheduling systems do not accurately estimate the impact of the rescheduling solution on future schedule execution in the real world.
6. Predictive-reactive approaches are the most effective ones when exceptions occur with low frequencies.

# Chapter 4 Work program

## 4.1 Introduction

The work program described in this chapter concludes the state of the art of chapter 3. A combined simulation and optimization based system is to be developed using the predictive-reactive approach as applicable for the PMFS problem. The work program of this thesis is divided in two parts: predictive scheduling and reactive scheduling. The next chapter describes the concepts and the work done in more details.

## 4.2 Work program

In the predictive system, in order of importance, the following components are to be developed:

1. Develop a method to combine simulation and optimization for predictive scheduling, using the software package *eM-Plant* to serve as a starting point by considering some broader production system elements such as buffers.
2. Develop a method to consider fixed and flexible job routings with constraints on delivery times, such that the lateness is as less as possible.
3. Develop a method to consider machine maintenance times, tools, materials and resource availability times.
4. Develop a method to make sure the predictive schedule can be executed in the real world without creating bottlenecks.
5. Develop a method to consider fixed job routings without constraints on delivery times.
6. Do the above to work in *eM-Plant* simulation software.

In the reactive system, in order of importance, the following components are to be developed:

1. Develop a method to combine simulation and optimization for reactive scheduling using the software package *eM-Plant* to include some broader system elements such as buffers with limited capacities.
2. Develop a method to consider Adaptation Synchrony in the rescheduling computations.
3. Develop a method to estimate the impact of rescheduling on future

schedule execution in terms of problems such as bottlenecks, and if problems occur,

- a. Detect the problems, and solve them in the rescheduling step.
4. Develop a method to bring back deviations to their planned trajectories on exceptions, resulting in additional benefits on performance indicators.
5. Develop a method to reschedule as late as possible in the planning period resulting in additional benefits on performance indicators.
6. Do all of the above to work simultaneously in *eM-Plant* simulation software.

### **4.3 Conclusions**

In this chapter we have discussed the detailed work program which is related to most of the conclusions of the state of the art chapter. In the next chapter, the concepts of each of the points mentioned in the work program are described in much details along with algorithm and system development. Especially in connection to the points mentioned here, from the predictive part, points 1 to 5 are described and developed in chapter 5, while the implementation of the predictive system is done in chapter 7. In the reactive part, points 1 to 5 are described and developed in the next chapter, while the implementation is shown in chapter 7. Chapter 6 shows how all the work fits in together as a black box.

# Chapter 5 Concepts and solutions

## 5.1 Introduction

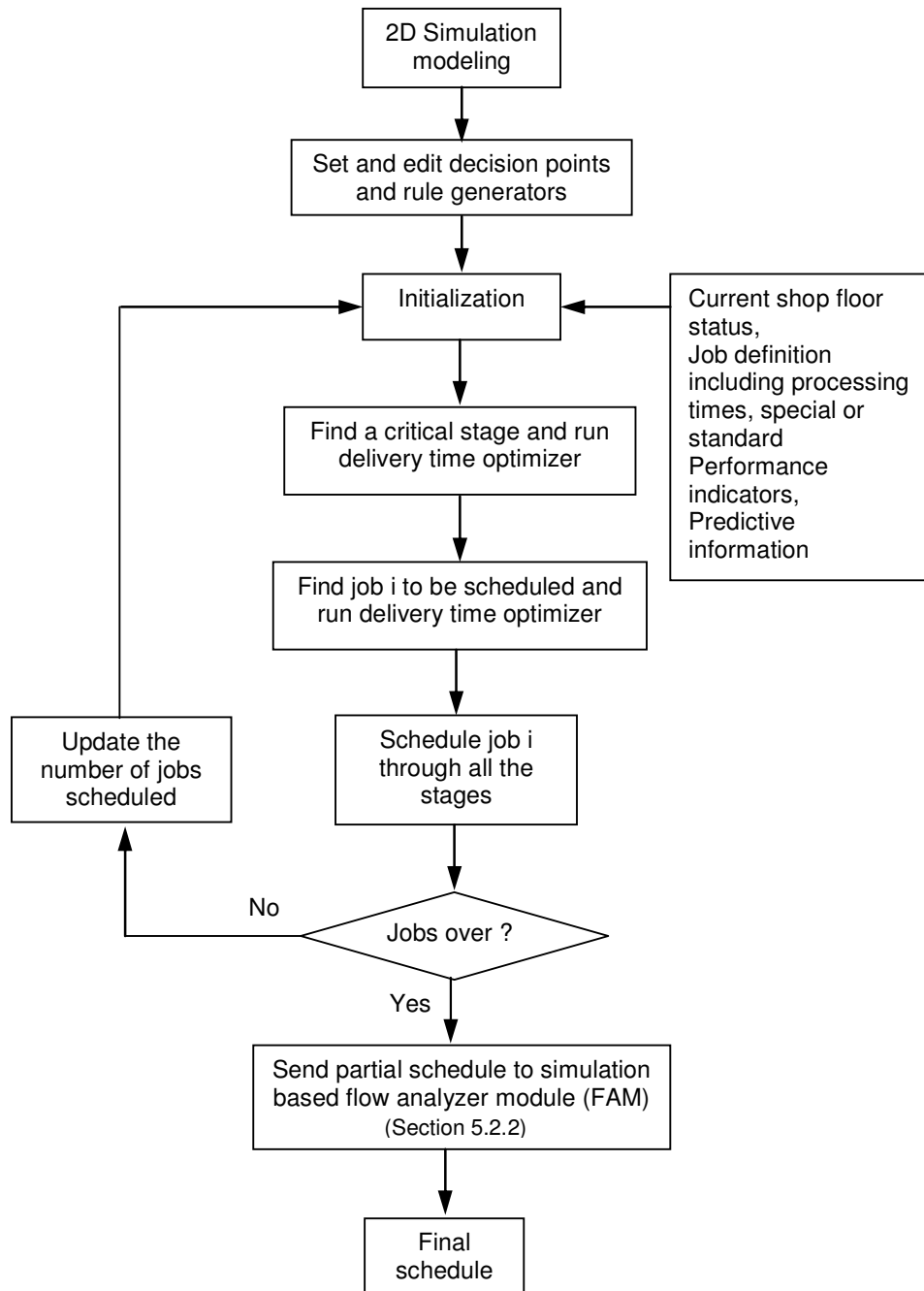
In this chapter, we describe the concepts and solutions development of the predictive and reactive systems. The configuration used is the parallel machine flow shop, with objectives and aims described in the earlier chapters. The predictive part of the algorithm is developed as a combination of optimization and simulation. The reactive algorithm is also based on combination of optimization and simulation. In the next chapter, a total integration of the system and the synthesis is described based on this chapter.

## 5.2 Concept and solutions for simulation and optimization assisted predictive scheduling

In the predictive part, a feasible production schedule for the next planning horizon (the next shift for instance) is generated. Figure 5.1 shows how the simulation and optimization system for predictive scheduling is organized. The first step is the 2D modeling phase, where the production system is modeled with the machines and stages, using *eM-Plant* simulation software along with data for the optimization. Then the user models and sets all the decision points (explained next) which are required for the rule-based simulation. The optimization algorithms are then run, which generate a rough plan. These algorithms consider details such as jobs routing and delivery time requirements along with details of tools, materials, and machine maintenance schedules. The algorithm runs by scheduling jobs through the flow shop one by one, by intermittently computing the bottleneck or critical stage (a stage which needs extra care when scheduling a job to reduce the make-span), and the delivery time optimizer, and considering the constraints, thus providing a rough plan. The algorithm is divided in two steps, the first one is to determine a critical stage for scheduling the standard jobs, then followed by determining which job needs to be scheduled when depending on the job type – which is standard job and special jobs. This second step is done by the delivery time optimizer. This way, standard jobs are scheduled according to the critical stage calculation, while special jobs are scheduled according to their required delivery times.

This partial plan is fed to the simulation based Flow Analyzer Module (FAM) for a detailed analysis. Simulation offers a possibility to model all the detailed elements of the production system thus fine tuning the plan obtained by the

optimization algorithm by checking and reacting to specific conditions set by the user. In addition it detects system blockages (bottlenecks) caused due to system

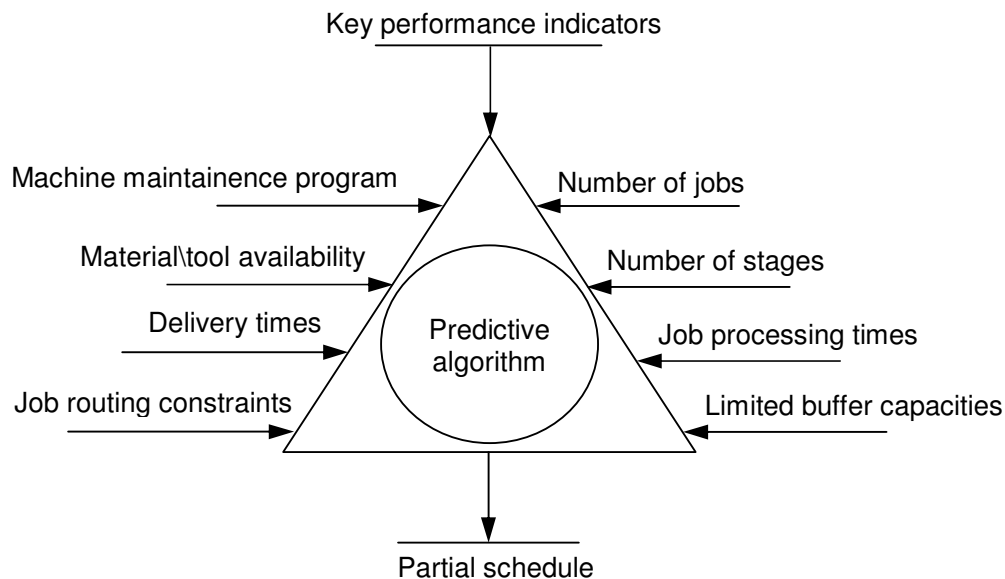


**Figure 5.1 Overall schema of the predictive scheduling system**

constraints like buffer sizing. Besides this, simulating a system with all detailed elements can give us the actual performance measures. The FAM uses the decision points (a point where a routing decision is taken in the model by a job) and the rule generators (rule generators contain the code for analysing local situations to generate a rule) within them to analyze conditions locally during the run-time of the simulation for broader system elements such as buffers with limited capacities. The system works in two parts, the optimization algorithm obtains a rough plan and the second part further refines the solution by locally over-riding the results of the rough plan according to the criteria set by the user. In the next sections the optimization algorithm is described in details, followed by describing the simulation based FAM unit and how it is combined with the optimization system.

### 5.2.1 Optimization algorithm

In Figure 5.2, the elements which are required as input to the predictive algorithm such as details like constraints on delivery times, tooling, material, equipment availability, are shown.



**Figure 5.2 Inputs and outputs to the predictive algorithm**

#### 5.2.1.1 Initialization

An example is taken to explain the steps one by one. Table 5.1 shows the input data to the algorithm consisting of jobs, processing times on each stage, tool, material and delivery time requirements. The configuration is a 2 stage flexible flow shop with 2 machines on each stage and 10 jobs. Machine 1 on stage 2 is set for

maintenance from 1:30:00.0000 to 2:00:00.0000 (30 minute duration). The times are set with the *eM-Plant* time format H:M:S.MS (Hours: Minutes: Seconds. Milliseconds). In each step of the first iteration, the calculations are done by hand and are described. For each of the later iterations, details and results are given in Appendix 1 for the reader to refer to.

**Table 5.1 Example to explain algorithm**

| Jobs           | Stage 1 | Stage 2 | Path | Delivery time | Material\ Tool availability | Machine Maintenance  |
|----------------|---------|---------|------|---------------|-----------------------------|--|
| 1              | 25      | 20      |      |               |                             | Machine 1 stage 2 not available from 1:30:00.0000 to 2:00:00.00-00 |
| 2              | 25      | 20      |      |               |                             |  |
| 3              | 25      | 25      |      |               |                             |  |
| 4              | 30      | 20      |      |               |                             |  |
| 5              | 15      | 15      |      | 2:30:00.000   | 1:00:00.000                 |  |
| 6              | 30      | 20      | 1,1  |               | 30:00.000                   |  |
| 7              | 10      | 20      | 2,1  | 2:00:00.000   |                             |  |
| 8              | 15      | 30      | 1,2  |               |                             |  |
| 9              | 30      | 15      | 2,1  |               |                             |  |
| 10             | 15      | 15      | 1,1  |               |                             |  |
| $\sum p_{i,j}$ | 220     | 200     |      |               |                             |  |

Step 1 Derive predictive machine maintenance schedules.

If schedule exists,

a. Get all machines which are maintained,

$$k_j^m \text{ for } \forall (k=\{1,2,\dots,m_j\} \text{ and } j=1,2,3,\dots,J).$$

b. For each machine  $k_j^m$ , get unavailable start and ending times as  $k_{tSt}$  and  $k_{tEnd}$ . In the current example,  $k_j^m$  would be machine 1 on stage 2.

$$\text{So } k_{tSt} = 1:30:00.0000 \text{ and } k_{tEnd} = 2:00:00.0000.$$

else, go to step 2.

Step 2 Derive raw material availability schedule.

If schedule exists,

a. Get all jobs  $I$  which have schedule  $\forall (I \rightarrow I)$ .

b. For each such job in  $I$ , get the material available times as  $i_t$ .

Else, go to step 3. In the current example, for job 6,  $i_t = 30:00.0000$ .



Step 3 Derive tool availability schedule.

If schedule exists,

Get all tools  $t_{tNum}$  for which we have set availability for all unscheduled jobs,  $\forall (I \rightarrow I')$ .

a. Get each tool available times as  $t_{tAv}^i$  for  $\forall t_{tNum}$ .

Else, go to step 4.

Here it is assumed that the tools are placed in the respective machines, and that each job requires a number of tools from among the tools in the machine's magazine. In this step, we set the fact if some tools will be available after some time for each of the jobs. In the current example, for job 5,  $t_{tAv}^i = 1:00:00.0000$ .

Step 4 Derive special job schedule.

If schedule exists,

a. Get special jobs  $i_{Sp}$  for  $\forall (i_{Sp} \rightarrow I')$ .

b. Get special jobs each with job routing description  $R = (k_{j=1}^m, \dots, k_{j=J}^m)$  where  $k =$  one of  $\{1, 2, \dots, m_j\}$  and  $j = \{1, 2, 3, \dots, J\}$ , and with delivery times  $d_t$ . Arrange jobs in an increasing order of their delivery times. In the current example, this is shown as List 1 in Table 5.2. Note the job routings are shown as integers (2,1), meaning job 7 goes to machine 2 on stage 1 and to machine 1 on stage 2.

c. Get special jobs each with only delivery time  $d_t$  and arrange jobs in increasing order of their delivery times. In the current example, this is shown as List 2 in Table 5.3.

d. Get special jobs with only routing description  $R$ . In the current example job 6, 8, 9 and 10 are the jobs in this category. This is shown in Table 5.4.

Else, go to step 5.

**Table 5.2 List 1 (L = 1) of special unscheduled jobs with routing and delivery times**

| Jobs | Routing description R | Processing time on Stage 1 | Processing time on Stage 2 | Delivery time |
|------|-----------------------|----------------------------|----------------------------|---------------|
| 7    | 2,1                   | 10                         | 20                         | 2:00:00.0000  |

**Table 5.3 List 2 (L= 2) of special unscheduled jobs with only delivery times**

| Jobs | Processing time on Stage 1 | Processing time on Stage 2 | Delivery time |
|------|----------------------------|----------------------------|---------------|
| 5    | 15                         | 15                         | 2:30:00.0000  |

**Table 5.4 List 3 (L = 3) of special jobs with only routing description**

| Jobs | Routing description R | Processing time on Stage 1 | Processing time on Stage 2 |
|------|-----------------------|----------------------------|----------------------------|
| 6    | 1,1                   | 30                         | 20                         |
| 8    | 1,2                   | 15                         | 30                         |
| 9    | 2,1                   | 30                         | 15                         |
| 10   | 1,1                   | 15                         | 15                         |

**Table 5.5 List 4 (L = 4) of standard unscheduled jobs**

| Jobs | Processing time on Stage 1 | Processing time on Stage 2 |
|------|----------------------------|----------------------------|
| 1    | 25                         | 20                         |
| 2    | 25                         | 20                         |
| 3    | 25                         | 25                         |
| 4    | 30                         | 20                         |

In this step we only determine the jobs with special requirements on delivery time and job routing within the production network. For the purposes of scheduling, four lists are created as shown below with Table 5.5 showing List 4 with all standard jobs.

Step 5 Set  $I_o = \Phi$ . Get processing times ( $p_{i,j}$ ) at each stage and job ready times at stage-1 ( $a_{i,1}$ ) for all jobs to be scheduled ( $I$ ). Include the job ready times for jobs for which material is available later (step 2 = true), which will ultimately be allowed to start (be ready) at a later time. Include job ready times for jobs for which the tools will be available later (step 3 = true), which will ultimately be allowed to start at a later time. This is shown for the current example in Table 5.6.

**Table 5.6 Job ready times at stage – 1 ( $a_{i,1}$ )**

|           | Jobs |   |   |   |              |            |   |   |   |    |
|-----------|------|---|---|---|--------------|------------|---|---|---|----|
|           | 1    | 2 | 3 | 4 | 5            | 6          | 7 | 8 | 9 | 10 |
| $a_{i,1}$ | 0    | 0 | 0 | 0 | 1:00:00.0000 | 30:00.0000 | 0 | 0 | 0 | 0  |

Step 6 Calculate tails for each job at each stage, using the formula  $q_{i,j} = \sum_{j=m+1}^J p_{i,j}$ .

Tails are computed to find out the amount of work left at a certain stage. This is shown in Table 5.7 as tails for each job at each stage  $q_{i,j}$ .

**Table 5.7 Tails for each job at each stage**

| Jobs | Tails                  |                        |
|------|------------------------|------------------------|
|      | at Stage 1 - $q_{i,j}$ | at Stage 2 - $q_{i,j}$ |
| 1    | 20                     | 0                      |
| 2    | 20                     | 0                      |
| 3    | 25                     | 0                      |
| 4    | 20                     | 0                      |
| 5    | 15                     | 0                      |
| 6    | 20                     | 0                      |
| 7    | 20                     | 0                      |
| 8    | 30                     | 0                      |
| 9    | 15                     | 0                      |
| 10   | 15                     | 0                      |

**5.2.1.2 Detail procedure to handle deterministic events and delivery constraints**

Step 7 Calculate the earliest start times on all the stages as follows:

- Get the machine ready times  $a_{j,k}^m$  for all  $\{(j=1,2,\dots,J), (k=1,2,\dots,m_j)\}$
- Set the earliest start times ( $s_{i,t}$ ) at stage – 1 and downstream stages respectively for the jobs yet to be schedule as follows:

$$s_{i,1} = \max\{(a_{i,1}), \min\{a_{i,k}^m \mid k=1,2,\dots,m_1\}\} \quad \forall i \rightarrow I' \text{ and}$$

$$s_{i,j} = \max\{(s_{i,j-1} + p_{i,j-1}), \min\{a_{j,k}^m \mid k=1,2,\dots,m_j\}\} \quad \forall (i \rightarrow I', \text{ and } j=2,3,\dots,J).$$

This will depend on when the current jobs  $i'$  on stages ( $s_{i,t}$ ) and ( $s_{i,u}$ ) are finished indicated by  $a_{i,k}^m$  and  $a_{j,k}^m$  (machines). The maximum value is selected amongst the job available times and the earliest machine available times. Table 5.8 shows the earliest start times at stage 1 for all jobs and similarly table 5.9 shows the earliest start times at stage 2.

**Table 5.8 Earliest start times at stage 1 for all jobs**

| Jobs | $s_{i,1} \rightarrow$ Stage 1 |
|------|-------------------------------|
| 1    | 0                             |

|    |              |
|----|--------------|
| 2  | 0            |
| 3  | 0            |
| 4  | 0            |
| 5  | 1:00:00.0000 |
| 6  | 30:00.0000   |
| 7  | 0            |
| 8  | 0            |
| 9  | 0            |
| 10 | 0            |

**Table 5.9 Earliest start times at stage 2 for all jobs**

| <b>Jobs</b> | <b><math>s_{i,2} \rightarrow</math> Stage 2</b> |
|-------------|---|
| 1           | 25  |
| 2           | 25  |
| 3           | 25  |
| 4           | 30  |
| 5           | 75  |
| 6           | 60  |
| 7           | 10  |
| 8           | 15  |
| 9           | 30  |
| 10          | 15  |

Step 8 Calculate the critical stage among all the stages. A critical stage is one, which calls for extra care to be taken to schedule a job. So we find critical stage, and schedule a job such that the resulting make-span is as less as possible. The critical stage is found by determining the lower bounds similar to the ones used by Carlier (1987) and Gupta and Ruiz-Torres (2000) as used by Phadnis et. al (2001). Note that the critical stage computation will include the effects of the material, tools, machine unavailability's, and the special jobs obtained through the job ready times and the machine ready times. Once we get the critical stage, we select a job to be scheduled in the next step. For a detailed description of how the critical stage is calculated, refer section 3.2.1.1. The results for this example are shown in Table 5.10 (since Lower Bound 2 (LB2) remains the same for each job on each stage, it is written only once in the LB2 columns). As per the method, the critical stage is Stage 1 because of the highest lower bounds.

**Table 5.10 Lower bounds computed for both stages**

| Stage 1       |               | Stage 2       |               |
|---------------|---------------|---------------|---------------|
| Lower Bound 1 | Lower Bound 2 | Lower Bound 1 | Lower Bound 2 |
| 45            | 125           | 45            | 112.5         |
| 45            |               | 45            |               |
| 50            |               | 50            |               |
| 50            |               | 50            |               |
| 90            |               | 90            |               |
| 80            |               | 80            |               |
| 30            |               | 30            |               |
| 45            |               | 45            |               |
| 45            |               | 45            |               |
| 45            |               | 45            |               |

Step 9 Find the job to be scheduled as follows:

Refer to Figure 5.3 for details of the scheduling procedure. From the list of special jobs with routing and delivery times (shown as List 1), or the list with special jobs with only delivery time (List 2), select a job  $i'$  with the earliest delivery time. In the example here,  $L = 1$ , and job 7 is selected for scheduling because it is required the earliest.

- a. At each stage ( $j$ ), find the earliest available machine  $m'$  or the machine  $m'$ , if the job has a routing description.

$$a_{j,m'}^m = a_{j,m'}^m + p_{j,m'} \quad \forall (k = \{1, 2, \dots, m_j\} \text{ and } j = \{1, 2, \dots, J\})$$

Schedule job  $i'$  at each machine  $m'$  horizontally. Since no jobs are scheduled yet, either Machine 1 or Machine 2 on Stage 1 can be used to schedule Job 7. Since Job 7 has routing requirements, it is routed according to its routing description of 2,1.

- b. Update the ready time for each machine  $m'$  as  $a_{j,m'}^m = a_{j,m'}^m + p_{i,j}$ .

If any of the machine contains predictive maintenance, and if the machine ready time falls between or is equal to  $k_{tSt}$  and  $k_{tEnd}$ , or is greater than  $k_{tEnd}$ , accordingly add the duration of the maintenance to the machine ready times, as:  $a_{j,m'}^m = a_{j,m'}^m + p_{i,j} + (k_{tEnd} - k_{tSt})$ .

**Table 5.11 Updated machine ready times after scheduling Job 7**

| Stage 1   |           | Stage 2   |           |
|-----------|-----------|-----------|-----------|
| Machine 1 | Machine 2 | Machine 1 | Machine 2 |
| 0         | 10        | 30        | 0         |

At this stage, the selected special job (job 7 in this example), is scheduled horizontally through all the stages, on the desired routing description or the earliest available machine at each stage. After scheduling this job, the ready times for the scheduled machines are updated as shown in Table 5.11 for Job 7.

Step 10 Update the number of scheduled jobs as  $I_o = I_o + \{i'\}$ . Here  $I_o = \text{Job 7}$ .

Step 11 Determine the job finishing times for job  $i'$  from the machine ready time of the machine on the last stage. This is explained with the current example.

a. If,  
     (Earliest time job 7 can start on stage 2 + processing time on stage 2)  $\geq$  (Required delivery time – Tolerance), keep job 7 in the list of already scheduled jobs.

Else,  
     Delete job  $i'$  from the jobs already scheduled list as  $I_o = I_o - \{i'\}$ , go to step 12. This deletion means that the job is too early to be scheduled in this step. So we keep the job  $i'$  in its original list 1.

For the current example, this comes out as:

$$(10 + 20) \geq 100 \text{ (or } 120 - 20)$$

$$30 \geq 100$$

The result is false, meaning now is not the time for job 7 to be scheduled. The comparison is only made with the time job 7 is expected to exit the system after stage 2. Hence as seen the time 30 is compared to the required delivery time minus the tolerance. The earliest starting times for job 7 on stage 2 is bigger of the time when job 7 finishes on stage 1 which is 10, and the time when job 7 can actually start after one of the machines becomes ready earliest on the second stage which is 0 (see table 5.11 for machine ready times). So the job 7 is not scheduled and the control now considers job 5 for a scheduling try in the next step.

Step 12 Determine the next job with the earlier delivery times from among List 1 and List 2. This job may or may not contain routing constraint depending on whether it exists in List 1 or List 2.

If there exists a job in List 2,  
     set  $L = 2$ , go to step 9.

else,  
If there exists a job in List 1,

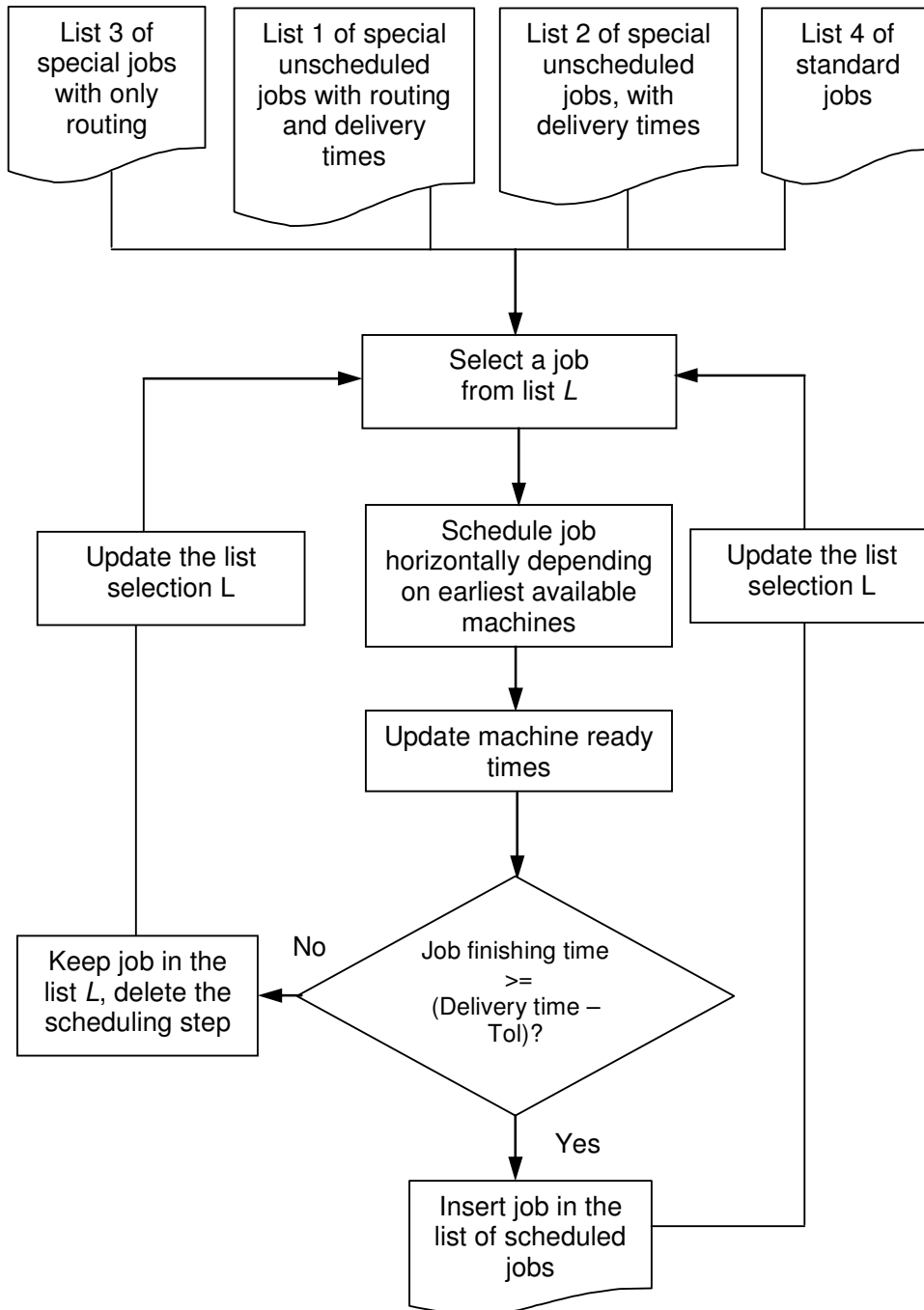


Figure 5.3: Accommodating delivery constraint for special jobs

set  $L = 1$ , go to step 9.

If no jobs exists in both lists 1 and 2,

set  $L = 4$ , go to step 13.

In this step, we determine the next job from List 1 or List 2 after trying scheduling the first special job, or if no more special jobs exist, we go on to step 13 for scheduling standard jobs. In the example, Job 5 is selected from List 2. After repeating iteration 9, 10 and 11, it is found that Job 5 too cannot be scheduled at this time point because it fails to meet the conditions set in step 11. Hence, step 13 is followed.

Step 13 Compute new job to be scheduled from the standard jobs list as follows:

- a. Set  $t = \min \{ a_{BN,k}^m \mid k=1,2,\dots,m_{BN} \}$

Let  $I''$  be the set of jobs not yet scheduled from the list 4 containing the standard jobs and available at time  $t$ .

$I'' \subseteq I'$ , where  $s_{i,BN} \leq t$

- b. If  $I_o = \Phi$ , then set  $t = \min \{ s_{i,BN} \}$ , where  $i \rightarrow I''$ . Go to step 13a.

In this step, the clock is set to the earliest possible start time for a job at the calculated critical stage. Then the heuristic, goes back to step 13a, since there could be more than one job. Jobs which are unavailable due to material or tooling problems will automatically be considered when time  $t$  is greater than the start job start times (or in other words when the jobs and tools become available). Note that the earliest possible start time for the critical stage includes machine unavailability's, if any.

- c. Select job  $i'$  with the longest tail on the critical stage:

$i' \rightarrow I''$  and  $q_{i',BN} \geq q_{i,BN} \forall i \rightarrow I''$ .

In the example, job 3 has the longest tail on the critical stage (Table 5.7). Hence this job is selected to be scheduled at this step. In case of a tie, select the job with the longest processing time at the bottleneck stage:

$p_{i',BN} \geq p_{i,BN} \forall i \rightarrow I''$ .

If there is still a tie, select a job with a lower number from the maximum processing times check.

Step 14 Schedule this new job  $i'$  depending on the result of the previous step, at all the stages:

- a. At each stage ( $j$ ), find the earliest available machine  $m'$ .

$a_{j,m'}^m = a_{j,m'}^m + p_{j,m'} \forall (k=\{1,2,\dots,m_j\} \text{ and } j=\{1,2,\dots,J\})$

Schedule job  $i'$  at each machine  $m'$ . Note that the earliest available machine is selected, which will ensure that the make-span is low and



and the machine idle time as low as possible.

b. Update the ready time for each machine  $m'$  as

$a_{j,m'}^m = a_{j,m'}^m + p_{i,j}$ . If any of the machine contains predictive maintenance, and if the machine ready time falls between or equals  $k_{tSt}$  and  $k_{tEnd}$ , or is greater than  $k_{tEnd}$ , accordingly add the duration of the maintenance to the machine ready times.

$$a_{j,m'}^m = a_{j,m'}^m + p_{i,j} + (k_{tEnd} - k_{tSt}) .$$

At this stage, the selected standard job, is scheduled through all the stages, on the earliest available machine at each stage respectively. After scheduling this job, the ready times for the scheduled machines are updated as shown in Table 5.12. Since in this case, the machine maintenances fall outside the processing times of this job, it does not affect the current calculation. Note that since this is the first job to be actually scheduled, and all machines have equal ready times, Machine 1 on both Stage 1 and 2 are selected by breaking ties according to lower numbered machines.

**Table 5.12 Updated machine ready times after scheduling Job 3**

| Stage 1   |           | Stage 2   |           |
|-----------|-----------|-----------|-----------|
| Machine 1 | Machine 2 | Machine 1 | Machine 2 |
| 25        | 0         | 50        | 0         |

Step 15 Update the number of scheduled jobs as  $I_o = I_o + \{i'\}$ . Here the  $I_o = \text{Job 3}$ .

Step 16 Go to step 8 and calculate the critical stage and scheduling process for the remaining standard as well as special jobs as per the procedure described. For the example taken, refer to Appendix 1 for detailed description and results obtained in each iteration. A description is given about what happens in each iteration. At the beginning of each iteration, the system is updated and so are the tables. At the end of each iteration, the jobs in the jobs already scheduled list is shown, building on the results of previous iterations.

Step 17 If any, schedule all the special jobs with no delivery time, but with only routing description. Since it is hard to determine when to insert these jobs in the scheduling process, these jobs are scheduled last after all standard jobs are scheduled. This is done to ensure that the overall makespan of

the system is still low in the presence of such jobs. In the current example, it is job 6, 8, 9 and 10 which is scheduled in this way.

Step 18 End.

In chapter 8, these approaches have been tested with more examples and variations and are found to produce good results.

### **5.2.2 Simulation based Flow Analyzer Module (FAM) for assessing predictive schedule**

In this section, we describe how to combine optimization and simulation to arrive at a final schedule. This combination will result in the consideration of the detailed system and a production schedule which will perhaps be better than the one computed by the optimization algorithm and also be free of problems (bottlenecks). Figure 5.4 shows the flow chart of how this system works. The step wise working of the system is described as follows, and the details are also described in the next section:

Step 1: Start and end simulation of the schedule computed by the predictive optimization algorithm.

Step 2: Record the data about job starting and ending times on each stage and makespan values obtained from the simulation run.

Step 3: Re-run the simulation with the Flow Analyzer Module (FAM) (explained in the next section) activated.

Step 4: During run-time, the FAM continuously analyze the situation at each decision point for each job. The decision points are the points where the rule generators are placed in the simulation model. The rule generators have the piece of code which analyse the local conditions, in this case, for a particular job for criteria such as avoiding bottlenecks, further optimization based on waiting times for jobs and checking the effects of details (maintenance schedules) considered by the optimization algorithm. The rule generators generate a rule if local conditions are fulfilled, by locally over-riding the result calculated by the optimization algorithm, only for a particular job. In the future this system can be extended to include more details of a production system such as pallets, material handling equipment, or other facts required to process certain jobs.

Step 5: Implement the new rule at the decision point by locally overriding the result

of the optimization algorithm, only for the job, machine and stage under

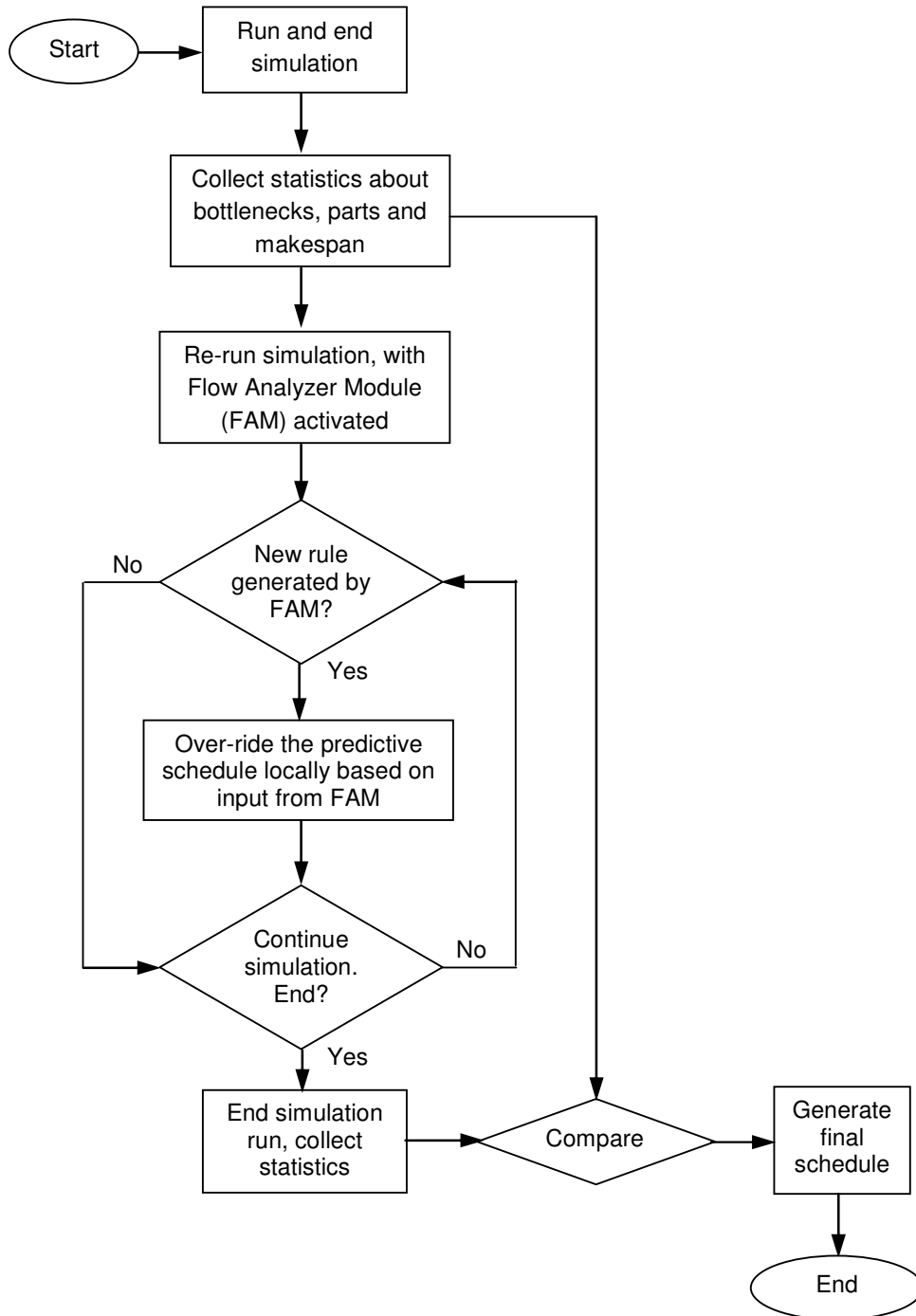


Figure 5.4 Simulation based Flow Analyzer (FA) to achieve validity

consideration. After executing the new rule, the jobs use the pre-calculated predictive schedule generated by the optimization algorithm for the next stage (before analyzing the situation on the next stage).

Step 6: Continue simulation. If required, go to step 4, else stop simulation. This way, all the jobs are analysed at the required decision points for criteria set by the user one by one. Collect the statistics on job starting and ending times on each stage and makespan.

Step 7: Generate a final schedule by comparing the result of the second simulation run (with FAM activated), and the schedule obtained by only simulating the result of the optimization algorithm.

### 5.2.2.1 The Flow Analyzer Module (FAM)

There are two situations the FAM module can consider which the optimization did not consider during its computations. In the first situation, the user knows about events which need to be considered during the simulation run. For instance, “Job 1 needs to wait for a special processing step of 10 minutes if it uses machine 2 on stage 2, and it needs 15 minutes if it uses machine 1 on stage 2”. Such information can be put into the rule generator which can analyze the situation during the simulation run to determine a rule. In the second situation, he has an overall knowledge of all events that can happen in his system, but does not know

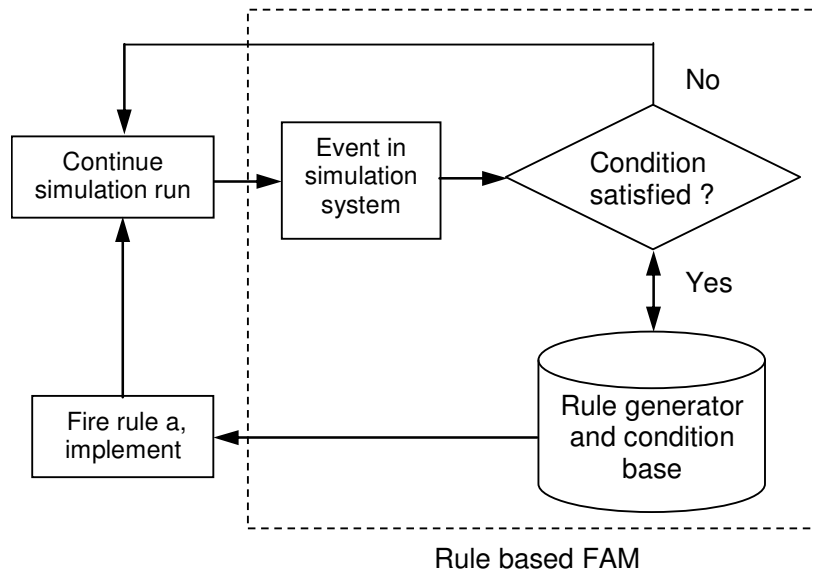


Figure 5.5 Concept of the simulation and rule based FAM

much about such events. For instance, he knows that a bottleneck can happen in the conveyor system, but may not be able to estimate its complete impact on the schedule. Such factual knowledge that the planner has obtained relevant to the problem can be realized as a database in the form of rule generators, conditions, and rules. This combination of simulation and optimization is a small step, which can be extended further. Figure 5.5 shows how the simulation based FAM works.

A general rule may be formulated as - if A then B, where A is a set of conditions on data and B is a set of instructions to be carried out if the rule is fired, using forward tracking principles. We use the forward tracking<sup>1</sup> principles because of the relative ease of developing and implementing such a system for the current production system configuration, instead of using backward tracking<sup>2</sup> principles where we can specify what conclusion we would like to reach, by specifying B.

Table 5.1 shows the interrelationships between decision points, conditions and rule generation. Section 5.2.2.1.1 describes rule generation for analyzing the flow for further optimality considering buffers and checking the impact of previous detail calculations of the optimization algorithm, while section 5.2.2.1.2 handles the condition of avoiding bottlenecks considering buffers and checking the impact of previous detail calculations of the optimization algorithm.

**Table 5.13: Interrelationships between decision points and rule generators**

| <b>Current condition / Rule generator</b> | <b>Condition A</b> | <b>Condition B</b> | <b>Condition C</b> |
|---|--------------------|--------------------|--------------------|
| Rule generator x                          | Instruction a      | ...                | ...                |
| Rule generator y                          | ....               | Instruction b      | ....               |
| Rule generator z                          | ....               | ....               | Instruction c      |

### 5.2.2.1.1 Rule generation for handling optimality

In this section a rule generation method for checking if the results and detail calculations of the optimization algorithm are optimal by handling buffers as production system elements is discussed. A small example is taken here to describe how this works in the simulation system in details. Table 5.14 shows the jobs and processing times on a 2 stage, 2 machine each production system configuration. Table 5.16 shows a sample rough schedule (paths for each job) obtained from the optimization algorithm described in the previous section, and Table 5.15 shows the buffer capacities for the case. A snap shot description of what exactly happens is seen in Figure 5.6, where job 1 is about to enter buffer 1 on its

<sup>1</sup> In Artificial Intelligence (AI), a form of reasoning that starts with what is known about the data and works toward finding a solution, Russel (2003).

<sup>2</sup> In AI, a form of reasoning that starts with what conclusion\goal is to be achieved and works backwards. The goal is broken into many sub-goals or sub-subgoals which can be solved more easily, Russel (2003).

**Table 5.14 Example jobs and system size**

| Jobs | Processing time on Stage 1 | Processing time on Stage 2 | Scheduled machine maintenance                         |
|------|----------------------------|----------------------------|---|
| 1    | 20                         | 10                         | Machine 1 on stage 2 not available from time 30 to 40 |
| 2    | 35                         | 20                         |   |
| 3    | 40                         | 15                         |   |
| 4    | 15                         | 15                         |   |
| 5    | 30                         | 40                         |   |
| 6    | 30                         | 35                         |   |
| 7    | 10                         | 30                         |   |

**Table 5.15 Buffer capacities on stage 2**

| Machine | Buffer capacity $Bf_c$ |
|---------|------------------------|
| 1       | 3                      |
| 2       | 2                      |
| 3       | 3                      |

**Table 5.16 Job routing according to optimization algorithm**

| Jobs | Machine on Stage 1 | Machine on Stage 2 |
|------|--------------------|--------------------|
| 1    | Machine 1          | Machine 1          |
| 2    | Machine 2          | Machine 2          |
| 3    | Machine 1          | Machine 1          |
| 4    | Machine 2          | Machine 1          |
| 5    | Machine 1          | Machine 3          |
| 6    | Machine 2          | Machine 3          |
| 7    | Machine 2          | Machine 1          |

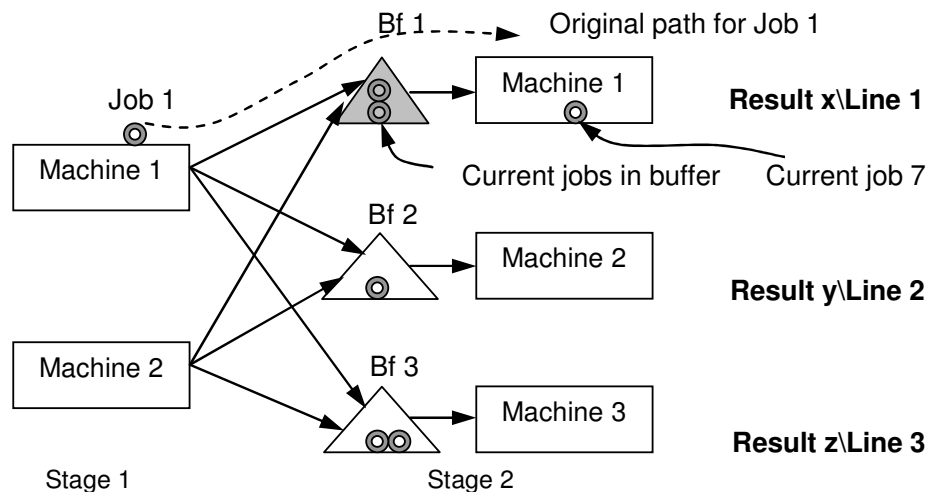
way to machine 1 at stage 2. At this moment, we analyze the number of jobs which are currently being processed on the next stage on each machine, including the number of jobs currently held in each buffer on the next stage ( $Bf_1$ ,  $Bf_2$ , etc).

It will use the following steps:

1. Get job  $i$  which wants to enter the next stage  $j$ , via a buffer. In this case, say job 1 is to enter stage 2 via buffer  $Bf_1$ , and is originally scheduled to enter machine 1 on stage 2 (see table 5.16 for the rough schedule).
20. Get the number of jobs in each buffer on stage  $j$ . Get the jobs  $i_c \subseteq I_o$  currently processed on machines  $m_j$  on stage  $j$ . For the current examples, job 3 and 4

are contained in buffer Bf 1, job 2 is contained in buffer Bf 2, and job 5 and 6 are contained in buffer Bf 3. In addition, job 7 is being processed on machine 1 on stage 2.

21. Compute the expected time  $t_m$  when each of the  $m_j$  machines on stage  $j$  become free after processing the current jobs  $i_c \subseteq I_o$ . For the current example, there is only job 7 being processed on machine 1 (see Figure 5.6), and no jobs on machine 2 and machine 3. Assuming job 7 is half way through its processing on machine 1, stage 2, so  $t_m = 15$  minutes for machine 1, and 0 for machine 2 and machine 3.
22. Compute the summation of processing times of all jobs on stage  $j$  in each buffer as  $\sum_{i=1}^{Bf_{CN}} (p_{i,j}^{Bf})$ . In the current examples, the summation comes out as 30 minutes in buffer 1, 20 minutes in buffer 2 and 75 minutes in buffer 3.
23. Sum up, for each of the alternative routes that job  $i$  can take on stage  $j$  (including the one which the optimization algorithm calculated), the machine free times, the processing times required for all jobs in each corresponding buffer and the processing time for job  $i$  on stage  $j$  as  $[ t_m + \sum_{i=1}^{Bf_{CN}} (p_{i,j}^{Bf}) + p_{i,j} ]$ .



**Figure 5.6 Snap shot to describe events for optimality and/or validity rule generation**

Here we have calculated the amount of time required when the job will have finished processing on each machine. In the current example, this will result in a time, when job 1 will leave stage 2 after processing itself on each one of the

machine on stage 2, and after all jobs (in each buffer and corresponding machines) are completely processed. For the current example this would result in values 55, 30 and 85 if job is processed on machine 1, 2 or 3 respectively – the time when the machines would be free again completely.

24. Compute if predictive machine maintenance has been scheduled at around the time the job  $i$  will possible finish on each of the machines on stage  $j$ , as follows:

If the time the machines would be free again falls between or equals  $k_{tSt}$  and  $k_{tEnd}$  or is greater than  $k_{tEnd}$ , accordingly add the duration of the maintenance to the machine free times as:

$$\text{The time required to process job } i = \left[ t_m + \sum_{i=1}^{Bf_{CN}} (p_{i,j}^{Bf}) + p_{i,j} \right] + (k_{tEnd} - k_{tSt}).$$

Else,

$$\text{The time required to process job } i = \left[ t_m + \sum_{i=1}^{Bf_{CN}} (p_{i,j}^{Bf}) + p_{i,j} \right].$$

If the start of maintenance time is less than the time each machine becomes free again, then the maintenance duration is added to the time the job  $i$  is expected to finish. On the other hand if the start of maintenance time is more than the time each machine becomes free again, then this means the machine is maintained at a later point in time. Accordingly for each situation the time required to process job  $i$  is computed. In this example, machine 1 on stage 2 is maintained from time 30 to 40 minutes. Since this falls within the time machine 1 will be free completely (if job 1 is processed on this machine), the maintenance duration is added to the machine free time as:  $55 + 10 = 65$ .

25. The result of step 6 will be in terms of a time unit, which will reflect the earliest time job  $i$  will finish after each of the machines (if processed on any). Let  $x$  be the result for the buffer Bf1\Machine 1 (Line 1),  $y$  be the result for Bf2\Machine 2 (Line 2) and  $z$  be the result for Bf3\Machine 3 (Line 3) as seen in Figure 5.6 and in Table 5.17.

Compute the differences with respect to  $x$  as follows:

$$x - y = a.$$

For this example, this comes out as,  $a = 65 - 30 = 35$ .

If  $a$  is positive, then this means job  $i$  could be processed faster by time  $a$ , if placed on Bf 2 (which had result  $y$ ).

If  $a$  is negative, then job  $i$  could be delayed by time  $a$ , if placed on Bf 2.

Similarly,

$$x - z = b.$$

For this example, this comes out as,  $b = 65 - 85 = -20$ .

Similar logic is applied for  $b$ .



26. Get buffer with maximum positive difference a or b, and get remaining capacity  $Bf_{cc}$  of this buffer. In this example, buffer 2 has remaining capacity of 1. When maximum positive value does not exist, solution cannot be more optimal than that of the optimization algorithm, go to step 9.

If  $Bf_{cc} \geq 1$

Fire rule and schedule job  $i$  on this buffer (line) by overriding the solution computed by the optimization algorithm.

In this example, this condition is true, and hence job 1 is scheduled on buffer 2 to be processed on machine 2, by overriding the result of the optimization algorithm, according to which job 1 had to be processed on machine 1.

Else,

Select the buffer with the next lower maximum positive difference, and if one exists, repeat step 8, else go to step 9.

27. End.

**Table 5.17 Earliest job  $i$  finishes on each stage**

| Value | Earliest time | Difference with respect to original path |
|-------|---------------|--|
| x     | 65            |  |
| y     | 30            | 35                                       |
| z     | 85            | -20                                      |

**Table 5.18 New job routing obtained after running simulation based FAM**

| Jobs | Machine on Stage 1 | Machine on Stage 2 |
|------|--------------------|--------------------|
| 1    | Machine 1          | Machine 2          |

Table 5.18 shows the new job routing for the snap shot example taken here. After this procedure the simulation again proceeds to do the same analysis with other remaining jobs throughout the model, at each and every decision point. Note that at the end of the analysis, the predictive machine maintenance times, tools, material availabilities, special jobs will also be considered by the entire system, as the optimization algorithm calculated when the jobs which have such constraints should be released in the system at the input. Note that the simulation based FAM system considers standard and special jobs **with delivery times only** and tries to finish these jobs as early as possible, so it does not affect the delivery time calculations for special jobs with only delivery times. All other special jobs are not

considered in these calculations because of their contrasting demands. Appendix 1.1 describes similar detailed hand calculations (where the above also becomes clear) for the example (Table 5.1) taken to describe the optimization algorithm calculation. Each instantaneous event data in the simulation model is described with results and screenshots. In the next section, we describe the method of generating rules for the case of bottlenecks.

### 5.2.2.1.2 Rule generation for handling schedule validity by avoiding bottlenecks

This rule generator will analyze local situations and avoid bottlenecks due to buffer sizing and check the detail calculations of the optimization algorithms. It is designed such that before a bottleneck is resolved during the simulation run, optimality is also considered. The way the rough schedule of the optimization algorithm is over-ridden for a bottleneck case is explained here using Figure 5.6, with data on processing times and buffer sizes shown in Table 5.19, Table 5.20, while Table 5.21 shows a sample schedule calculated by the optimization algorithm.

**Table 5.19 Example jobs and system size**

| Jobs | Processing time on Stage 1 | Processing time on Stage 2 | Scheduled machine maintenance                         |
|------|----------------------------|----------------------------|---|
| 1    | 20                         | 10                         | Machine 1 on stage 2 not available from time 30 to 40 |
| 2    | 35                         | 20                         |   |
| 3    | 40                         | 15                         |   |
| 4    | 15                         | 15                         |   |
| 5    | 30                         | 40                         |   |
| 6    | 30                         | 35                         |   |
| 7    | 10                         | 30                         |   |

**Table 5.20 Buffer capacities on stage 2**

| Machine | Buffer capacity $Bf_c$ |
|---------|------------------------|
| 1       | 2                      |
| 2       | 1                      |
| 3       | 3                      |

**Table 5.21 Job routing according to optimization algorithm**

| Jobs | Machine on Stage 1 | Machine on Stage 2 |
|------|--------------------|--------------------|
| 1    | Machine 1          | Machine 1          |

|   |           |           |
|---|-----------|-----------|
| 2 | Machine 2 | Machine 2 |
| 3 | Machine 1 | Machine 1 |
| 4 | Machine 2 | Machine 1 |
| 5 | Machine 1 | Machine 3 |
| 6 | Machine 2 | Machine 3 |
| 7 | Machine 2 | Machine 1 |

As seen in Figure 5.6 if buffer Bf1 is full (see Table 5.20 for buffer capacities in this example) and job 1 has to travel to Machine 1 (on stage 2) according to the pre-calculated schedule, then obviously there is a bottleneck situation which needs to be resolved as much as possible. To resolve this situation, we first compute the time when each machine will be free from machining its current job by undertaking the following steps:

1. Get job  $i$  which wants to enter the next stage  $j$ , via the blocked buffer. In this example, job 1 wants to enter stage 2 via buffer Bf1, and is originally scheduled to enter machine 1 on stage 2 (see Table 5.21 for the rough schedule).
2. Get the number of jobs in each buffer on stage  $j$ . Get the jobs  $i_c \subseteq I_o$  currently processed on machines  $m_j$  on stage  $j$ . For the current example, job 3 and 4 are contained in buffer Bf 1, job 2 is contained in buffer Bf 2, and job 5 and 6 are contained in buffer Bf 3. In addition, job 7 is being processed on machine 1 on stage 2.
3. Compute the expected time  $t_m$  when each of the  $m_j$  machines on stage  $j$  becomes free after processing the current jobs  $i_c \subseteq I_o$ . For the current example, there is only job 7 being processed on machine 1 (see Figure 5.6), and no jobs on machine 2 and machine 3. Assuming job 7 is half way through its processing on machine 1, stage 2, so  $t_m = 15$  minutes for machine 1, and 0 minutes for machine 2 and machine 3.
4. Compute the summation of processing times of all jobs on stage  $j$  in each buffer as  $\sum_{i=1}^{Bf_{ON}} (p_{i,j}^{Bf})$ . In the current examples, the summation comes out as 30 minutes in buffer 1, 20 minutes in buffer 2 and 75 minutes in buffer 3.
5. Sum up, for each of the alternative routes that job  $i$  can take on stage  $j$  (including the one which the optimization algorithm calculated), the machine free times, the processing times required for all jobs in each corresponding buffer and the processing time for job  $i$  on stage  $j$  as  $[ t_m + \sum_{i=1}^{Bf_{ON}} (p_{i,j}^{Bf}) + p_{i,j} ]$ .

Here we have calculated the amount of time required when the job will have

finished processing on each machine. In the current example, this will result in a time, when job 1 will leave stage 2 after processing itself on each one of the machine on stage 2, and after all jobs (in each buffer and corresponding machines) are completely processed. For the current example, this would result in values 55, 30 and 85 if job is processed on machine 1, 2 or 3 respectively – the time time when the machines would be free again completely.

6. Compute if predictive machine maintenance has been scheduled at around the time the job  $i$  will possibly finish on each of the machines on stage  $j$ , as follows:

If the time the machines would be free again falls between or equals  $k_{tSt}$  and  $k_{tEnd}$  or is greater than  $k_{tEnd}$ , accordingly add the duration of the maintenance to the machine free times as:

$$\text{The time required to process job } i = \left[ t_m + \sum_{j=1}^{Bf_{CN}} (p_{i,j}^{Bf}) + p_{i,j} \right] + (k_{tEnd} - k_{tSt}).$$

Else,

$$\text{The time required to process job } i = \left[ t_m + \sum_{j=1}^{Bf_{CN}} (p_{i,j}^{Bf}) + p_{i,j} \right].$$

If the start of maintenance time is less than the time each machine becomes free again, then the maintenance duration is added to the time the job  $i$  is expected to finish. On the other hand if the start of maintenance time is more than the time each machine becomes free again, then this means the machine is maintained at a later point in time. Accordingly for each situation the time required to process job  $i$  is computed. In this example, machine 1 on stage 2 is maintained from time 30 to 40 minutes. Since this falls within the time machine 1 will be free completely (if job 1 is processed on this machine), the maintenance duration is added to the machine free time as:  $55 + 10 = 65$ .

7. The result of step 6 will be in terms of a time unit, which will reflect the earliest time job  $i$  will finish after each of the machines (if processed on any). Let  $x$  be the result for the buffer Bf1\Machine 1 (Line 1),  $y$  be the result for Bf2\Machine 2 (Line 2) and  $z$  be the result for Bf3\Machine 3 (Line 3) as seen in Figure 5.6 and in Table 5.22.

Compute the differences with respect to  $x$  as follows:

$$x - y = a.$$

For this example, this comes out as,  $a = 65 - 30 = 35$ .

If  $a$  is positive, then this means job  $i$  could be processed faster by time  $a$ , if placed on Bf 2 (which had result  $y$ ).

If  $a$  is negative, then job  $i$  could be delayed by time  $a$ , if placed on Bf 2.

Similarly,

$$x - z = b.$$

For this example, this comes out as,  $b = 65 - 85 = -20$ .

Similar logic is applied for b.

8. Get buffer with maximum positive difference a or b, and get remaining capacity  $Bf_{cc}$  of this buffer. When maximum positive value does not exist, select buffer with minimum negative value from a or b. This is because if maximum positive difference exists, then a better makespan can also be achieved by scheduling on this buffer (besides resolving the bottleneck situation, by overriding the optimization result), while if buffer with minimum negative value is selected, then a worse makespan but a bottleneck free situation will result.

If  $Bf_{cc} \geq 1$ ,

Fire rule and schedule job  $i$  on buffer. Go to step 9.

Else,

Select the buffer with the next lower maximum positive difference if it exists, or the one with minimum negative value if one exists and repeat step 8, else go to step 9. In this step it is checked if a bottleneck free situation can be created on other buffers. If not, this means, the solution cannot be improved by overriding the results of the optimization algorithm.

In the present example, the buffer with maximum positive difference leads to buffer 2. Then it is checked if the balance capacity on buffer 2 is atleast 1. This check fails, and then the next buffer with minimum negative value is chosen, which is buffer 3 (note that if this check was successful, a better makespan by 35 minutes as well as a bottleneck free situation would have resulted). It is checked if this buffer has balance capacity. This check is successful, and job 1 is scheduled to buffer 3 to be processed on machine 3. This will result in a makespan longer by 20 minutes, but a bottleneck free situation – the goal of this rule generator.

9. End procedure.

Table 5.23 shows the result of the FAM analysis for the snap shot example of Figure 5.6.

**Table 5.22 Earliest job  $i$  finishes on each stage**

| Value | Earliest time | Difference with respect to original path |
|-------|---------------|--|
| x     | 65            |  |
| y     | 30            | 35 (a)                                   |
| z     | 85            | -20 (b)                                  |

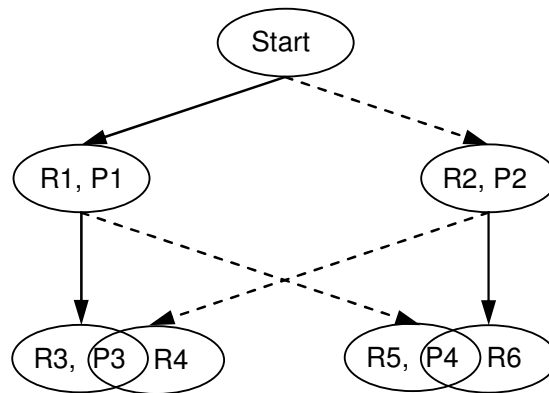
**Table 5.23 New job routing obtained after running simulation based FAM**

| Jobs | Machine on Stage 1 | Machine on Stage 2 |
|------|--------------------|--------------------|
| 1    | Machine 1          | Machine 3          |

As explained earlier, the simulation then continues and generates similar rules on the occurrence of another bottleneck elsewhere during the simulation run. This rule generator may result in some special jobs delivered later than that calculated by the optimization algorithm because of the fact that it allows jobs to finish later, but assures a bottleneck free situation, as was the case in the example.

**5.2.2.2 Sequential rule firing and its consequences**

In this section, the detailed aspects of the interaction between the initial schedule and the schedule obtained after analysis using the FAM are discussed in terms of rules generated by the latter. It may happen as a result of the simulation based overriding system, that two rules are fired for two jobs sequentially directed towards one position in the model, may not be in the same sequence as calculated by the optimization algorithm.

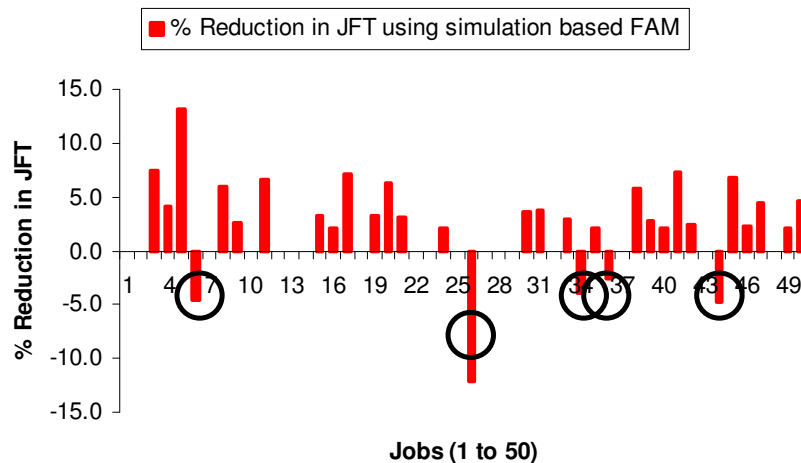


**Figure 5.7 Sequential rule generation**

This can happen due to the varying processing times of the jobs considered and the complex interactions between several jobs and positions. Figure 5.7 shows the phenomenon. As seen rule R1 is generated first to be implemented at position 1 in the model as is shown by thick arrowed line. Rule 2 is generated after rule 1, hence shown in dotted line, and implemented at position 2. In the next step, rule 3 is generated and implemented at position 3, but soon afterwards, rule 4 is generated and implemented at same position 3. However it was precalculated by the predictive system that a particular job should have gone to position 4 instead of

position 3. This is a conflict situation and as a result rule 4 will override rule 3 and continue with the simulation. No special handling is required for tackling such sequentially generated and implemented rules. This is because at the end of analyzing all the schedules generated by the system, the user can still select the best overall schedule.

To demonstrate the effect, a sample result is shown in Figure 5.8 (more results and data used for such results and tests are described in the results chapter). The figure shows on the y-axis the percentage reduction of Job Finishing Times (JFT) and x-axis the number of jobs in the system. The percentage reduction is calculated by using the data on Job Finishing Times of the initial schedule (calculated by the optimization algorithm and then simulation), and the data on Job Finishing Times of the new schedule (obtained by the FAM). It can be seen that some jobs finish later than the initial schedule using the FAM analysis (shown by the circles). This is due to the phenomenon described earlier – the phenomenon that some jobs are overridden by other jobs in the flow analysis step when they are actually in a different sequence prior to starting the flow analysis.



**Figure 5.8 Consequences of sequential rule generation**

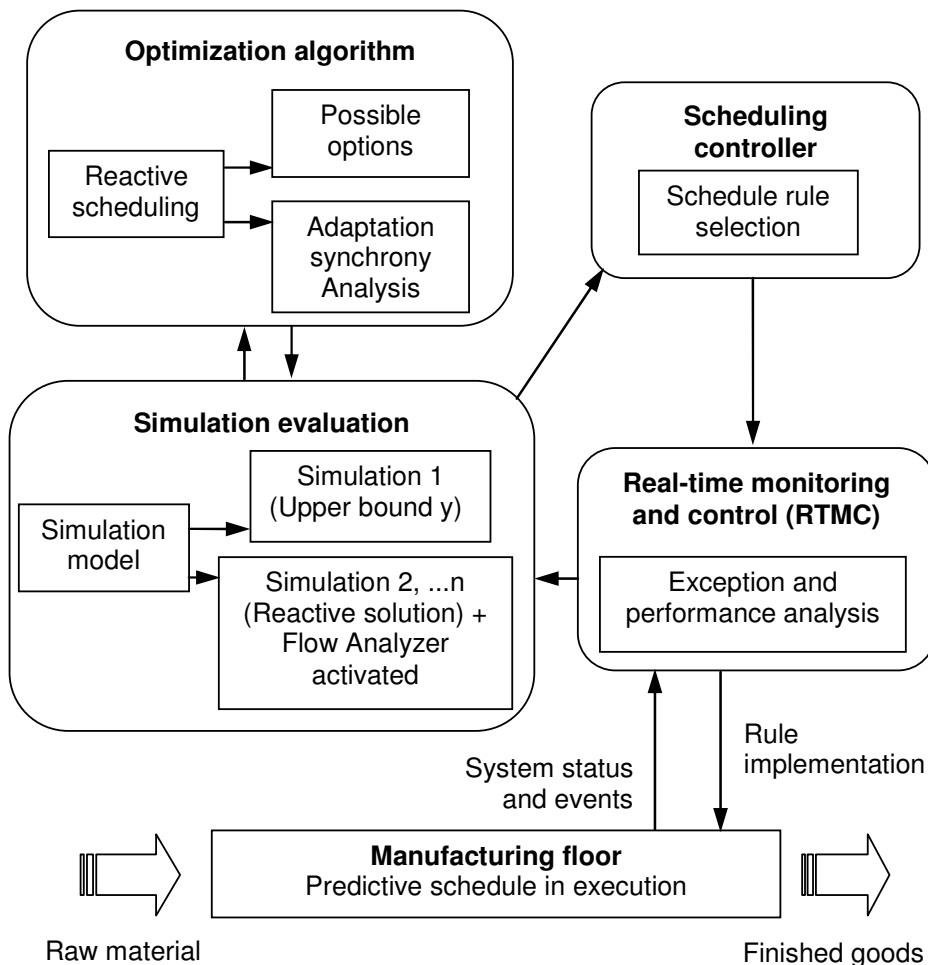
### 5.3 Concepts and solutions for reactive scheduling

According to the overall concept explained earlier, after the user evaluates the predictive schedule in the previous shift, he executes a final schedule on the shop floor real world in the following shift. Note that the predictive schedule computed the period before considers broader elements of the production system like buffers and details like materials, machine availability, delivery times etc and this schedule has already been evaluated and analyzed using the simulation based

FAM in the earlier period. Exceptions occur rarely during the execution of the schedule on the shop floor, and when they do, the reactive system provides solutions using a combination of simulation and optimization. Figure 5.9 shows the overall concept of the system. The following are the components of the simulation and optimization assisted reactive system:

**1. Real-time monitoring and control**

This component receives events from the shop floor, and periodically monitors performance of the shop floor. It also sends all scheduling controller information to the shop floor and dispatches instructions accordingly. The sub-component within the RTMC is the Exception and Performance Analysis.



**Figure 5.9 Simulation and optimization assisted reactive scheduling**



This checks all incoming events and invokes rescheduling on execution exceptions (criteria defined by users) with the remaining operations under the current shop floor conditions.

## **2. Simulation evaluation**

The simulation system captures the entire model of the physical manufacturing system and receives events from the RTMC making it run parallel to the real manufacturing system. This component simulates the entire system at least once. First, as soon as a reaction is desired, the simulation system simulates rapidly the effect of the deviation (simulation 1, Figure 5.9) and computes the upper bound (make-span) value  $y$  as seen in Figure 5.9 and Figure 5.10. The upper bound is the result of the total deviation from the original predictive schedule, if we choose not to do anything, until the end of the planning period. So the upper bound is essentially the worst-case scenario for the performance measure, which should not be exceeded (as much as possible) by the final solution of the reactive system. By simulating the entire system, we can be certain that the upper bound value  $y$  corresponds to the system that is modeled in a detailed way to consider some broader elements of a manufacturing system – something which is hard to do using a pure optimization algorithm. Second, the simulation evaluation component receives input about the better decision (options) computed by the optimization (re-scheduling) algorithm considering adaptation synchrony problems (when exactly to reschedule in the real world considering its evolution). These inputs from the optimization algorithm are simulated to achieve the required performance indicators selected by the user. Upon selection of a final rescheduling solution, this is evaluated with the Flow Analyzer Module (FAM) activated (shown as simulation 2,...,n + Flow analyzer activated in Figure 5.9) to check the validity of the better solution – that the solution does not create problems in the future due to rescheduling. The result of the last simulation run (i.e. with the FAM activated) should ideally give the make-span value lower than the upper bound value  $y$  computed earlier. If the opposite were true, this would mean, that it would be best not to re-schedule the current system. The functioning of the optimization algorithm is explained in details next.

## **3. Optimization component (Re-scheduling algorithm)**

After the simulation evaluation system computes the upper bound, the control is passed over to the optimization algorithm. This algorithm will compute the most probable rescheduling solutions alongwith guidelines on solving the Adaptation Synchrony problem and passes them back to the simulation evaluation system. The simulation system then evaluates each solution. The optimization algorithm basically further guides the simulation to a better solution considering

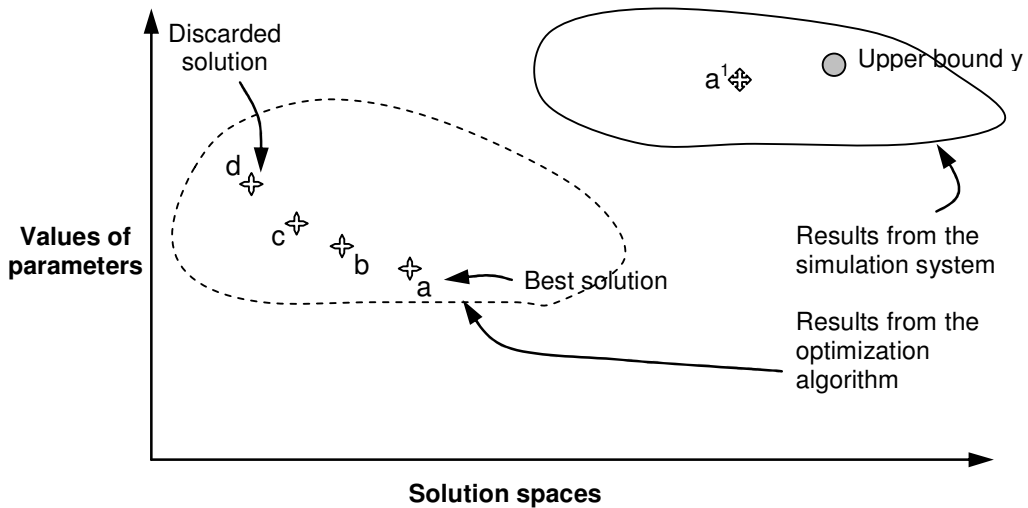
Adaptation Synchrony. In the end, the system presents a final rescheduling solution obtained for the optimization of a particular performance indicator. The optimization algorithms take two forms:

### 3.1 Match-up rescheduling algorithm

This algorithm tries to match up the schedule after deviation from its original trajectory. This is done by measuring each job starting time deviation and job sequence deviation, and accordingly developing a solution to bring back these deviations to much as possible to zero.

### 3.2 Selective rescheduling algorithm

This algorithm reschedules only a selected few jobs. This algorithm aims to fit in the jobs in the planning horizon by rescheduling selected jobs without addressing each job starting time deviation. This algorithm results in a system where deviations are done later in the planning period.



**Figure 5.10 Solution spaces and bounds**

Both these algorithms use some rough methods to generate good solutions shown as parameters a, b, c and d in Figure 5.10 (of the next decision to take). The optimization algorithm then gives these solutions to the simulation component which evaluates them in detail. It is necessary to use rough methods since with the pure algorithm it is hard to consider the all other details of the production system like buffers. Hence, we use simulation (combined with the Flow Analyzer) to simulate to evaluate the result of implementing the selected decisions (shown by a<sup>1</sup>

in Figure 5.10). This result is compared by the simulation system to the upper bound  $y$  obtained from simulating earlier as a feasibility check. In this way, the optimization algorithm does some rough calculations and reduces the solution space, while the simulation takes this space and determines the best solution. It should be noted that the optimization algorithm gives the most important solutions also considering Adaptation Synchrony, while the simulation system simulates to check if this solution can still fulfill the global upper bound needs, besides doing validity and synchrony checks.

#### 4. Scheduling controller

The scheduling controller gets input from the simulation evaluation after it obtains the results given to it by the optimization system. The scheduling controller passes on this result to the real time monitoring and analysis component, which further passes down this result to the real production system. Figure 5.11 shows the time line of the computations taking place in the system. The system functions as explained in the earlier section. On a time scale, as soon as an exception occurs at 8:00 am, the simulation system is activated based on user settings. This calculates the upper bound  $y$  of the system, and perhaps if further problems (such as bottlenecks) will happen at 8:45 am due to the exception.

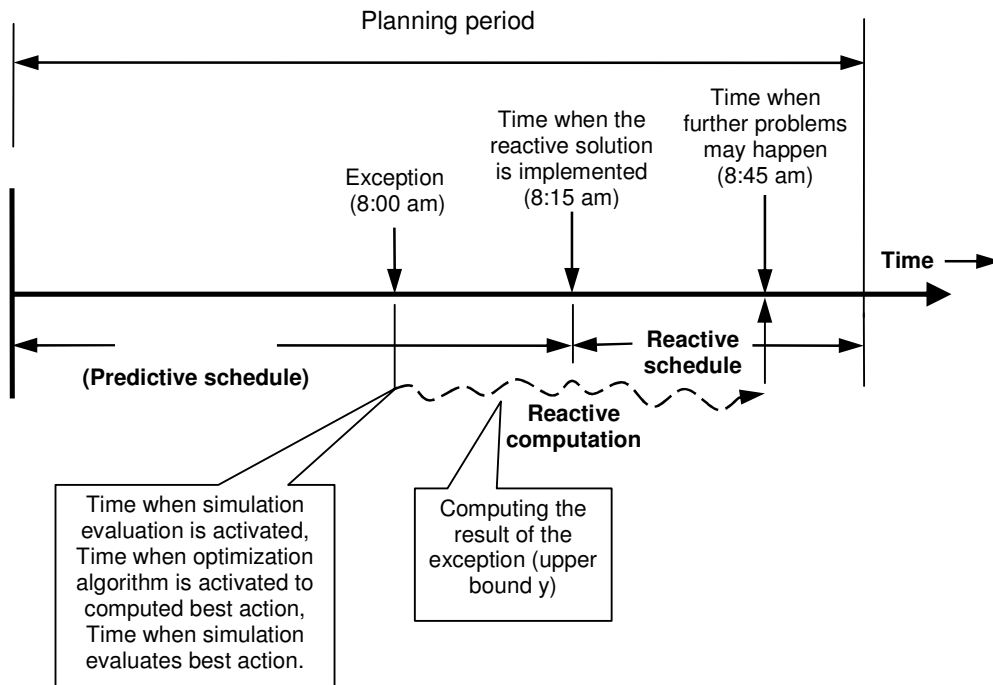


Figure 5.11 Time line of computations

Then the optimization algorithm is activated which computes the better candidate solutions. The simulation system once again, simulates these solutions. When the final solution is computed, it is finally decided that the reactive solution is implemented at 8:15 am, depending again on the user selected preferences and results of the Adaptation Synchrony.

### **5.3.1 Justifications for using these methods**

The following are the justifications for using the proposed methods for reactive scheduling:

1. A pure optimization algorithm will make it hard to compute accurately the result of an exception in the form of the upper bound, especially considering a parallel machine flow shop problem and additional elements such as buffers. Simulation gracefully captures all these details, and provides inputs to the optimization algorithm for developing possible rescheduling solutions.
2. A pure optimization algorithm will make it hard to compute the entire result of a rescheduling iteration, due to similar reasons mentioned in point 1. Using optimization to provide the better candidate solutions and rough solutions, and then using simulation cuts down computational efforts and increases accuracy than by using pure optimization.
2. Simulation provides a better way to assess if the solution (obtained from the rough optimization algorithm) we implement is really worth the efforts in the global system.

### **5.3.2 Matchup rescheduling for real-time control**

In this approach, the idea is to try as much as possible to come back to the original schedule upon the occurrence of an exception by making selective local changes to the schedule in such a way that performance indicators such as starting time deviations, sequence deviations for all jobs or the makespan are as close as possible to the original schedule. Figure 5.12 shows the rescheduling system.

As soon as an exception occurs, an upper bound is computed. The upper bound computation can be performed using an algorithm or a simulation system. In our case, since the system is quite complex (varying part processing times with multiple routings between stages and buffer elements), it is required to use simulation for the computation of the upper bound. The most important reason to use simulation is to be able to help in the accurate computation of the capacity that exists where rescheduling could possibly take place. Another reason to use simulation for the computation of the upper bound is due to the parallel machine system configuration – it is mostly the case that the upper bound is not equal to (but in fact is less than) the original predictive makespan plus the exception duration! This is due to the fact that with the exception some jobs can advance to the next

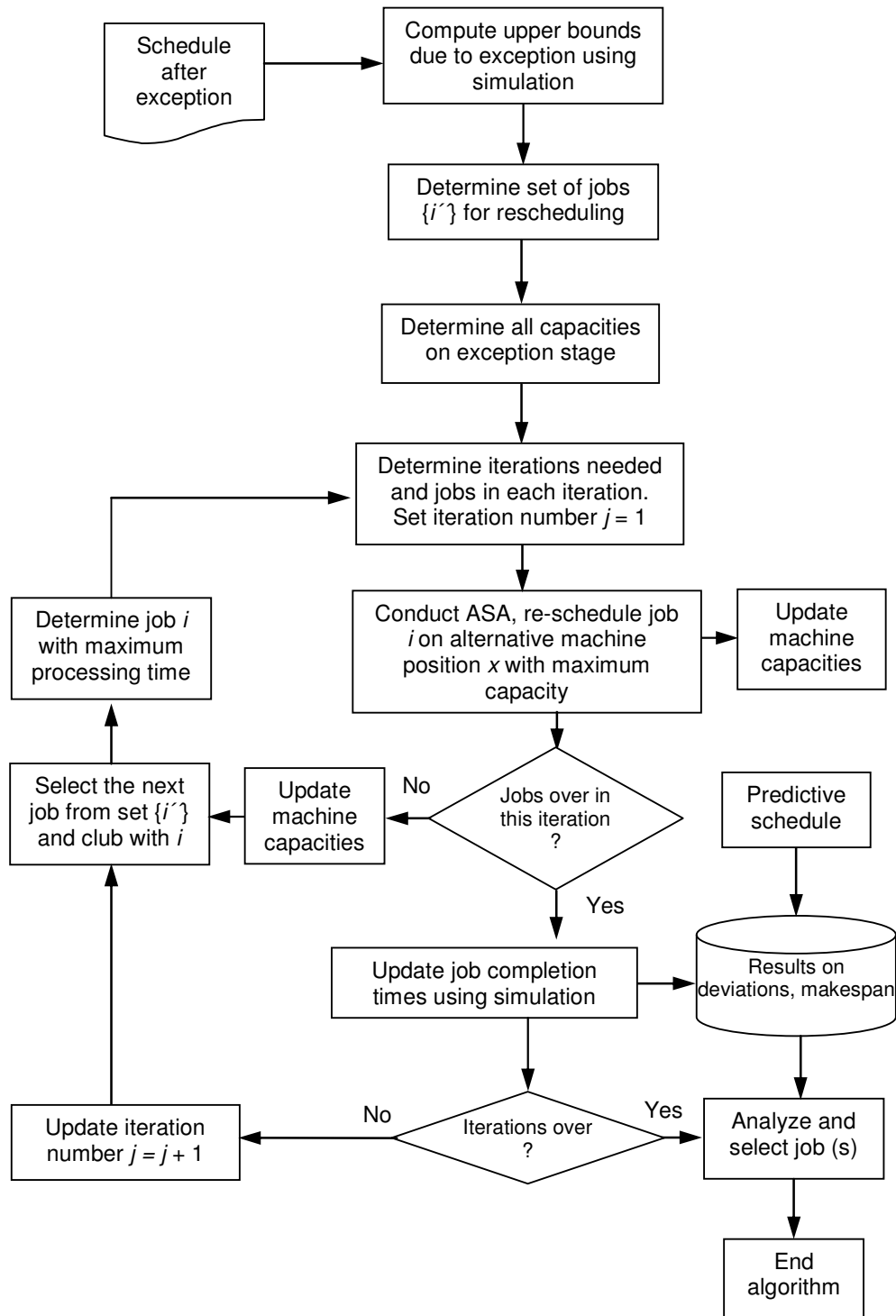


Figure 5.12 Matchup rescheduling for real-time control

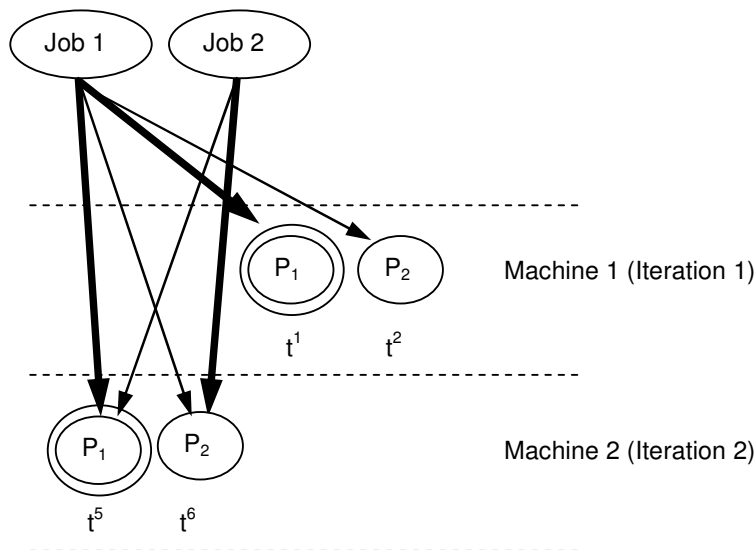
stage on the parallel machine configuration, thus reducing the upper bound makespan further. This upper bound represents the worst case scenario as a result of doing nothing, or in other words, the result of not re-scheduling. Then, on the same stage where the exception happened, we determine the amount of capacity that is available on all the alternative machines using a heuristic algorithm with input from the simulation run. A number of jobs are then created which are candidates for re-scheduling (the rescheduling set  $i''$ ). These jobs are the ones which should have started on the machine during the duration of the exception, according to the predictive schedule. For these candidate jobs, the total number of simulation iterations and the number of jobs within each iteration is computed and set. Jobs are rescheduled using an Updating Capacity Principle (UCP). In the first iteration, Job 1 is selected (from the set  $i''$ ) and is planned to be re-scheduled by the algorithm at a certain position  $P_1$  at time  $t^1$  on the alternative machine where maximum capacity exists (see Figure 5.13). An Adaptation Synchrony Analysis (ASA) is also conducted at this step, which attempts to solve co-ordination problems, which is described separately in the next section.

Since there is only one job in the first iteration, the system simulates this constellation and saves the results in a database. Following this, the algorithm updates the iteration number to the second iteration and gets the associated jobs with this iteration (jobs 1 and 2 from set  $i''$ ). From these two jobs, the one with maximum processing time on the stage where the exception occurred is selected. This job is then placed at a certain position at a machine with highest capacity. Note that at the beginning of each iteration, the inputs about the highest capacity are obtained from the upper bound simulation run. Again, just before conceptually rescheduling the ASA is conducted – explained in the next section. If there are more jobs in the iteration (true – second iteration has two jobs), the machine capacities are updated using a heuristic, new information on rough capacities are obtained and the second job is rescheduled on the newly calculated highest capacity machine (could be different than the one where job 1 was rescheduled). This is shown as an example in Figure 5.13.

At the end of this iteration, the jobs are placed at the calculated positions and a simulation run is carried out to save results on performance indicators on this run. If there are more iterations, the same process is repeated, where within each iteration, jobs are selected in the order of maximum processing times and machine capacities are updated intermittently. This method of updating capacity in our experience gives much better results than other methods we have developed and tested. A reason for this is that rescheduling jobs with maximum processing times first on machines with highest capacities seem to prioritize jobs which results in faster processing for jobs and balancing of loads on machines. This method of combining simulation and optimization results in fewer simulation runs (we do not

simulate each and every constellation), whilst still providing good results (shown later). Simulation is used to compute the exact maximum capacity for a system that can contain anything from different processing times on machine, buffers and transportation elements – something that can be extended further. The optimization algorithm then takes this as an input at the start of each iteration to keep the number of simulation iterations to a bare minimum.

As mentioned earlier, just before each sub-iteration and iteration, an Adaptation Synchrony Analysis (ASA) is conducted. This may alter the exact position (as calculated by the initial computations of the optimization algorithm) where a job may be rescheduled considering the future system state. This is explained in the next section.



**Figure 5.13 Tree of the iterations for positions and job rescheduling**

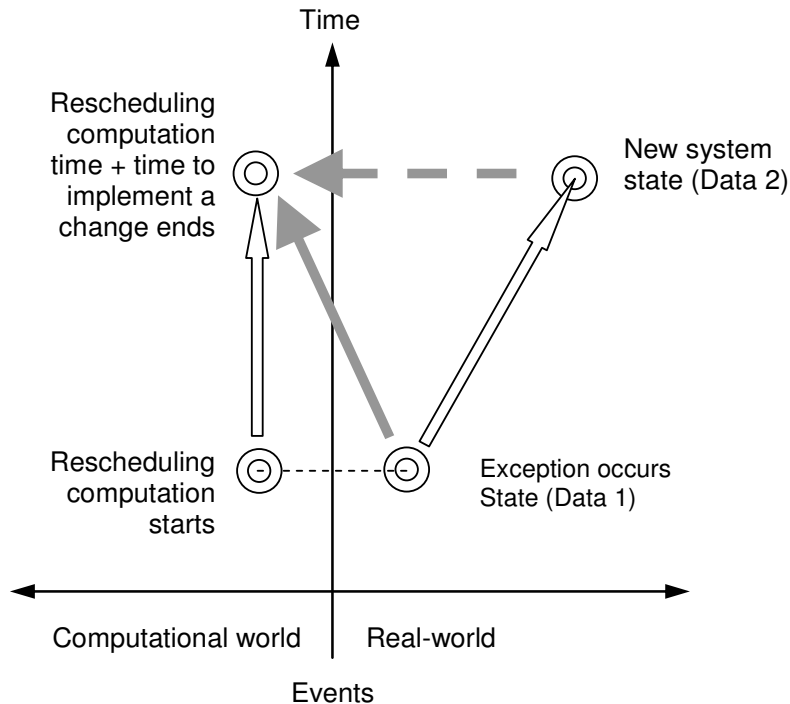
### 5.3.2.1 Adaptation synchrony analysis (ASA) during rescheduling

Adaptation synchrony defines how the actual process continues during its computation and modification. As seen in Figure 5.14, when the rescheduling computation starts, the data at the time of the exception was considered for computation. When the rescheduling computations and implementations would be finished (assuming we know or can determine how much time is needed to compute and implement a change), the real system state has evolved to a new state (corresponding to data 2). So the data used for the computation of a rescheduling solution corresponds to the wrong state of the system as marked by the solid grey arrow, with the specific result that the position where we wish to

reschedule a particular job may not be available, since a job around that position has already progressed ahead in time, making it infeasible to attempt the rescheduling step.

The following is the pseudo-code of the ASA system and the description of each point where necessary:

1. Let  $i''$  = number of jobs selected for rescheduling,
2. Let  $j$  be the machine selected for rescheduling,
3. Let  $P_i$  at time  $t^1$  be the position calculated by the optimization algorithm for job  $i$  (see Figure 5.16 for the illustration). This position is the earliest possible time a job can be rescheduled,
4. Get the job change implementation time  $t_{ch}$  from the “change chart(s)” for job  $i$ . The idea of change charts is described next.

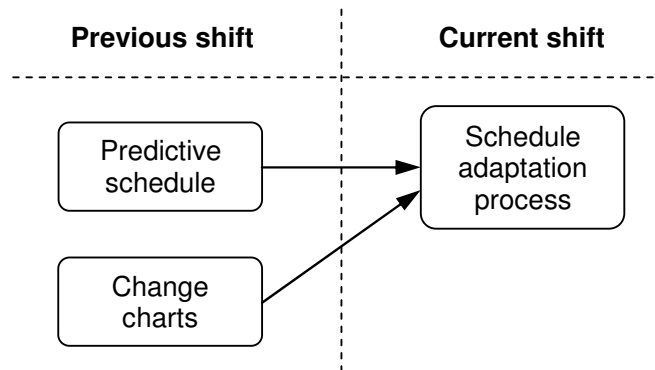


**Figure 5.14 Real-world state evolution and interaction with rescheduling**

The idea here is to include the computational times for the simulation runs, the computational times of the optimization algorithm, and most importantly the time required to actually make actual changes on the shop floor which would include times required for taking jobs out of their current position, times required for transportation and times required for setting the jobs in their new positions. Other actual times required for changes can be included in the system. The computation



times (iteration time using simulation) duration can be estimated to a high degree of accuracy as we have already carried out simulation runs earlier (the upper bound simulation for instance) from which we can determine the time required for a particular iteration or iterations. The times to actually reschedule a job from one machine to other on the shop floor can be estimated a priori (line operators can fill up a “change chart(s)” with relative ease for each job change on other machines on the same stage). The concept and the location where and how the change charts are to be filled and used is shown in Figure 5.15.



**Figure 5.15 Change charts filled up in the previous shift used in the current shift adaptation process**

This time will include additional times like set-up times, transportation times, etc in particular the following factors:

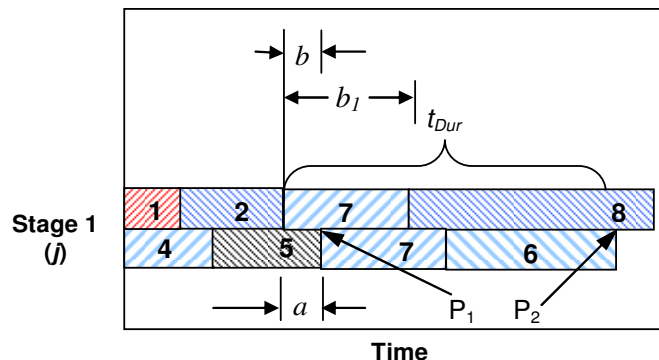
1. The time required by the system to first detect the exception and report it to the computer system (considered as zero-time factor during implementation because this has the same effect as the following factors).
2. The time required to dismantle the job, machine, or operator from the exception machine or resource (modeled as set-out times factor during implementation).
3. The time required to transport the job, machine and resources on alternative machines (modeled as transportation times factor during implementation).
4. The time required the job may have to wait until it can be set up on the new alternate machine (modeled as set-in times factor during implementation).
5. The time required to set-up the job on the new alternate machine.

All these times are added to result in a total time to be considered in this analysis.

5. Get the times required for the future simulation iterations as  $t_{Sim}$  and compute total time as  $t_{total} = t_{ch} + t_{Sim}$ . This  $t_{total}$  is shown as two example cases  $b$  and  $b_1$  in Figure 5.16. Note that  $t_{Sim}$  is the time estimated for the entire solution, while  $t_{ch}$  is the time required to actually change 1 particular job to its new position.

6. Compute the time  $t_{free}$  machine  $j$  (the one where job  $i$  is going to be rescheduled) will become free earliest after machining each job cumulatively to take in the rescheduled job  $i$ .  $t_{free}$  = remaining processing times for the job currently being processed on machine  $j$  + processing times for one or more of the remaining jobs on previous buffer of machine  $j$ .
7. If  $t_{total}$  (for example  $b_1$  in Figure 5.16)  $>$   $t^1$  (at  $P_1$ ), then
8. Compute new position  $P_{1\ new}$  with  $t_{1\ new} >$   $t_{total} (b_1)$  and  $t_{free}$  but with  $P_{1\ new}$  (position between one of the jobs that machine  $j$  has to machine) just after a job that machine  $j$  will machine at a later time period. Decide to proceed with simulation iteration on the new position.

To help understand points 5 to 8, consider the situation in Figure 5.16. Assume that the rescheduled job is job 7, which is shown at its new possible position.  $t_{Dur}$  is shown as the exception duration in the figure, while “ $a$ ” is shown as the time between the start of the exception and the start of position  $P_1$ . Now let’s assume that the change time (total time for computation and change) for job  $i$  is  $b_1$ , which is greater than the time “ $a$ ”. If this is the situation, obviously the simulation iteration with job 7 at position  $P_1$ , does not make sense because we will not be able to implement this solution (if we finally select it) in the real-world, because by the time we have a solution ready, job 6 would already have progressed into machining on the rescheduled machine according to its old schedule.



**Figure 5.16 Adaptation synchrony analysis computations**

In this situation, the next position  $P_2$  (through the computation of  $t_{free}$ ) is tried for rescheduling until the condition that the computational time for change is less than the subsequent positions of rescheduling. Note that job 6 is to originally start after job 5, before rescheduling.

On the other hand, if the change time for job  $i$  is  $b (< "a")$ , then obviously, we will have a possibility to implement this solution in the real world, as job 6 would not have started by then. When there are events where factors cannot be considered in

the total times, a provision is made in the simulation model to account for such events by scheduling them at a later – appropriate time. An example is the set-in time which starts after actually putting the job at its new position, i.e. after the duration of the total time is finished.

To describe the ASA system more, a comparison is made between the time the machine on which potential rescheduling is to take place becomes free and the total change time. If the change time exceeds the potential rescheduling position (each position also has a time), then obviously, we may not reschedule the job at that position. This is because by the time we reschedule this job physically the real system is ahead in time, consequently meaning that the potential position calculated by the optimization algorithm is no longer available. In such a situation it is then determined if the next position is available. If it is available, the job is rescheduled at the new position, thus overriding the initial position calculated by the optimization algorithm. The simulation iteration is then performed with the new position at the end of the iteration with the new positions calculated.

9 Else, decide to proceed with simulation iteration with position  $P_1$ .

10 If more jobs in set  $i''$ , select next job  $i+1$  and go to step 2, else, end iteration and simulate all jobs at the decided positions.

This way the system places a job at an appropriate position by computing in the future and by considering the scenario that the current (iterated) job(s) will be finally selected for rescheduling in the end after going through all iterations! After each iteration, the results on the starting times, make-span, sequence changes are saved in the database as explained earlier. After considering each iteration for re-scheduling, comparison is made between the predictive schedule and the result of re-scheduling the jobs at various iterations. Some job iterations may result in values of makespan higher than the predictive schedule, but result in lower starting time deviations with some sequence deviations. As a result, the user is presented with these results, in a consolidated form. The user then selects a particular rescheduling solution and does the post rescheduling analysis using the simulation based Flow Analyser Module (FAM), which is described in the next section. This post rescheduling analysis will reveal if the rescheduling solution will have problems in the future execution when actually implemented in the real-world. In this analysis, the FAM system will solve the potential future problems by over-riding as less jobs as possible and present a final solution. The final solution deviations and performance data are recorded and presented to the user. The final rescheduling solution is compared to the upper bound simulation run to find out if its worth to implement this solution in the real-world or not. The final solution is then given back to the scheduling controller and on to the RTMC module as seen and explained in Figure 5.9.

### 5.3.2.2 Detailed algorithm for match up rescheduling and the ASA system

In this section, the detailed algorithm is described with the help of an example. A 2x2 model was taken (2 machines on each of the 2 stages). The exception was set to occur from 25 to 65 minutes on machine 2 on stage 1. Table 5.24 shows the number of jobs used for explaining the example. The predictive schedule generated of this example is shown in Appendix 2, Figure A5 as a gantt chart. The exception would fall at the end of job 4 and continue until the middle of job 5, shown by solid dotted braces. The detailed steps in the system are mentioned as follows:

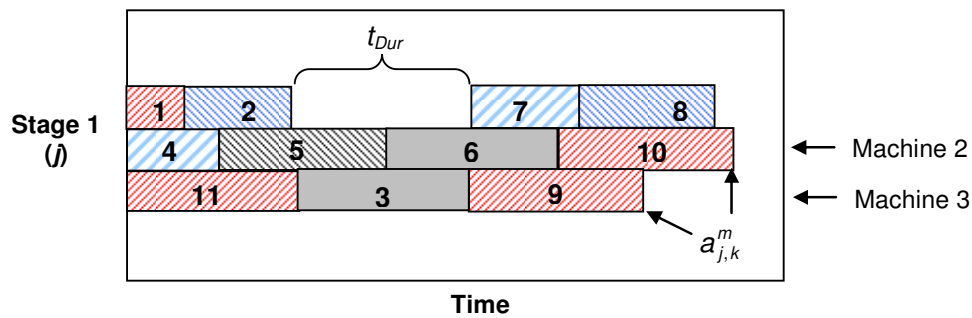
**Table 5.24 Example to explain algorithm**

| <b>Jobs</b> | <b>Stage 1<br/>Processing times<br/>(minutes)</b> | <b>Stage 2<br/>Processing times<br/>(minutes)</b> |
|-------------|---|---|
| 1           | 25  | 20  |
| 2           | 25  | 20  |
| 3           | 25  | 25  |
| 4           | 30  | 20  |
| 5           | 15  | 15  |
| 6           | 30  | 20  |
| 7           | 10  | 20  |

1. Get the predictive schedule  $Sch_{predictive}$ . In this schedule all the jobs are scheduled.
2. Get time  $t_D$  = time when exception happened, and duration of exception as  $t_{Dur}$ . In this example  $t_D = 25$  minutes, and  $t_{Dur} = 40$  minutes.
3. Compute the upper bounds using simulation as the worst-case situation by first determining the directly affected jobs, and the impact of these jobs on the indirectly affected jobs. In  $Sch_{predictive}$ , put all jobs on machine  $k$  at stage  $j$  after time  $t_D$  in set  $bd' = \{1, 2, 3, 4, \dots, n\}$ , where  $bd' \subseteq I_0$ . Note that  $I_0$  contains all the scheduled jobs. In the current example, the directly affected jobs are job 4, 6 and job 5. As a result of the directly affected jobs, there are other jobs which are indirectly affected in the schedule. In  $Sch_{predictive}$ , for job  $i = 1$  in  $bd'$  on stage  $j$ , calculate the new time when this job will finish as  $(t_{Dur} + p_{i,j})$ . For each job following (after) job 1 on stage  $j$ , update the job completion times using simulation. Because these jobs also exist on subsequent stages,  $j + 1, \dots, J$ , during the run-time of the simulation, the jobs starting and ending times are updated accordingly. In the end the result is completely updated schedule which will result in the upper bound make-span value  $make-span^{UB}$ . Save this result in

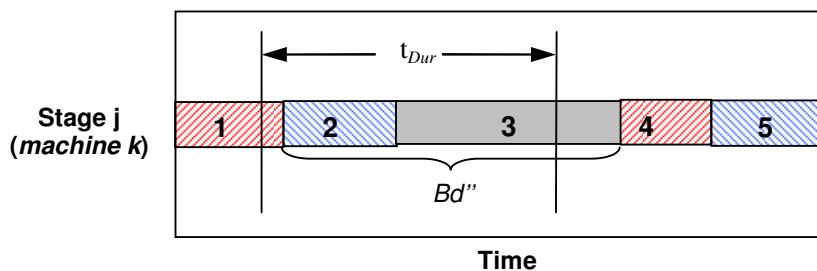
a database.

- Determine capacity on the exception stage by checking the earliest available time  $a_{j,k}^m$  for all the machines on this stage. The machine with the lowest earliest available time will have the highest capacity to absorb the exception, and will be the one where the job will be rescheduled to. Figure 5.17 shows the earliest available time  $a_{j,k}^m$ , on machines 2 and 3 for a generic example. In the current example, since there is only one machine on the exception stage,  $a_{j,k}^m = 85$  minutes (1:25:00.0000). This comes out as the time machine 1 on stage 1, becomes available earliest (can be calculated by adding up the processing times of all jobs on machine 1 on stage 1, and the idle times of the machine).



**Figure 5.17 Earliest available machines on exception stage**

- Create a new set of jobs  $bd'' \subset bd'$ . Jobs in set  $bd''$  are selected according to the principle shown in Figure 5.18. All jobs between the exception duration beyond the job on which exception happened and time when the job where the exception duration ends are selected. In the current example, job 6 and job 5 are in the set  $bd''$ .



**Figure 5.18 Determining jobs in the set of jobs to reschedule**

- Determine the number of iterations  $it$  needed and numbers of jobs within each

iteration and save this data in a table, shown for this example in Table 5.25. Set the number of jobs in each iteration equal to the iteration number starting with the first job in set  $bd''$ . So, if set  $bd''$  has 3 jobs, iteration 1 will have the first job, iteration 2 will have job 1 and 2, and iteration 3 will have job 1, 2 and 3. In the current example, job 6 goes into first iteration, and job 6 and job 5 go into the second iteration.

7. Compute the time required for the change by computing the time simulation runs depending on the total number of iterations, and determine the change times from the change charts. In this example, the estimated simulation time is roughly 2 seconds, and the change values were set to 5 minutes each for set-out, transportation, and set-in factors, so the total change time would be 15 minutes and 2 seconds. This change time is an input to the ASA conducted in the next step.

**Table 5.25 Iterations and jobs selected for rescheduling**

| <b>Iteration number</b> | <b>Jobs in iteration</b> |
|-------------------------|--------------------------|
| 1                       | Job 6                    |
| 2                       | Job 6, Job 5             |

**Table 5.26 Capacities on earliest available alternative machines**

| <b>Iteration</b> | <b>Earliest available machines time</b> |
|------------------|---|
| 1                | <b>85 minutes</b>                       |
| 2                |   |
| ...2.1           | <b>85 minutes</b>                       |
| ...2.2           | <b>115 minutes</b>                      |

8. Select the job  $i$  in iteration 1 from Table 5.25 and temporarily reschedule this job on a particular position  $P_1$  as shown in Figure 5.16. Conduct the ASA. As explained in the ASA section, this will result in Job 6 completely rescheduled at time 40 minutes and 2 seconds. This comes out as,  $t_b$  (25 minutes) + the change time (15 minutes and 2 seconds) = 40 minutes and 2 seconds. Note that job 1 on machine 1 will be over after 50 minutes (see Figure A5 in Appendix 2), meaning that the ASA did not have to make changes to the most probable position.
9. If there are no more jobs in this iteration (true – first iteration has only 1 job), update the job completion times using simulation. Save the data on make-span, starting times in a database.
10. If number of iterations  $it > 1$  (more than 1 iteration), update iteration number as

$it = it + 1$ , get the jobs for this iteration from the iterations table. In the second iteration, there is job 6 and job 5.

11. Select the job with the maximum processing times from among these jobs and reschedule this job on a particular position  $P_1$ . Job 6 has a processing time of 30, higher than job 5 and hence is selected for rescheduling first. Conduct ASA.
12. Update machine capacities after rescheduling this job by first updating the earliest available times on the machine where position  $P_1$  is located as:

$a_{j,k}^m = a_{j,k}^m + p_{i,j}$  where  $p_{i,j}$  is the processing time for the rescheduled job on the new machine. After rescheduling job 2,  $a_{j,k}^m = 85 + 30 = 115$  (1:55:00.0000).

13. Determine new capacity on the exception stage by checking the earliest available time  $a_{j,k}^m$  for all the machines on this stage. If there are more machines on the exception stage, then the new capacity is selected according to the principle described in Figure 5.17. The lowest value of the earliest available machine is the one with maximum capacity. In this example, there is only one machine, and hence is selected for rescheduling other jobs in this iteration. Table 5.26 shows the capacities obtained for all iterations for this example.
14. If there are more jobs in this iteration (true because  $it \neq 1$ ) select the next job with the next lower highest processing times go to step 11. In this step job 5 is selected for rescheduling, and in step 11 it is actually rescheduled by conducting the ASA. Job 5 is rescheduled at time 40 minutes and 2 seconds, as explained earlier as it had the same change times.
15. Simulate this iteration and store results on job starting times and makespan in the database.
16. If there exists more iterations, go to step 10. Else, go to step 17.
17. Consolidate the results and derive final rescheduling solution to implement based on user choice of performance indicator. The results are consolidated by computing for all stages the average starting time deviation, the sequence deviations for all jobs rescheduled from their original starting times (predictive schedule) as follows:

$$\text{Average starting time deviation on Stage } j = \frac{devSt^{Job1} + devSt^{Job2} + devSt^{Job3}}{n(\text{number of jobs})}$$

$$\text{Total starting time deviation for the total problem} = \frac{\text{Average starting time deviation Stage } j}{J(\text{Number of stages})}$$

The results of these calculations are shown in Appendix 2, Table A63 and Table A64.

18. From the selected solution, the FAM is used and a post rescheduling analysis is

carried out. After this step, the final result is compared to the simulation upper bound, and the user is presented the final solution.

19. End.

This small example is also solved with the developed system and the result is shown in Figure A6 and A7 in Appendix 2. The above system tries to minimize starting time deviations by trying to return to the original trajectory on the occurrence of the exception, whilst solving problem of Adaptation Synchrony Analysis. As seen in this particular example, the upper bounds and the selected rescheduling solution were the same, meaning that in this case, rescheduling does not improve the situation.

### **5.3.2.3 Post rescheduling analysis using the simulation based FAM**

Once a solution has been computed using the system, the user can carry out post rescheduling solution analysis (See Figure 5.19 for overview). This analysis basically checks if the rescheduling solution will cause disruptions (bottlenecks) in the real-world if implemented as calculated. It may suggest further changes to the rescheduling solution to avoid problems in the real-world due to the rescheduling action calculated by the system (described in the previous sections). How this fits in the systems described earlier is shown in Figure 5.9.

If the user selected to do this analysis, the system proceeds with a simulation run, and analyses the directly and indirectly affected jobs. If required, it over-rides the jobs from the newly computed schedule. A Flow Analyser Module (FAM) is implemented to conduct this analysis. The FAM is the bottleneck rule generator – the same as that used for the predictive schedule generation. There are decision points within the model, and within them, rule generators are placed. These rule generators have the code which analyze the situation locally for problems such as bottlenecks, and provide an alternative rule if certain conditions are fulfilled during the run-time of the simulation. The user can decide between a rescheduled plan with his performance measures perhaps with problems which will occur in the future or a rescheduled plan with better performance measures without problems in the future. Both these alternatives have implications on make-span, and optimization criteria as will be seen in the results section of this thesis.

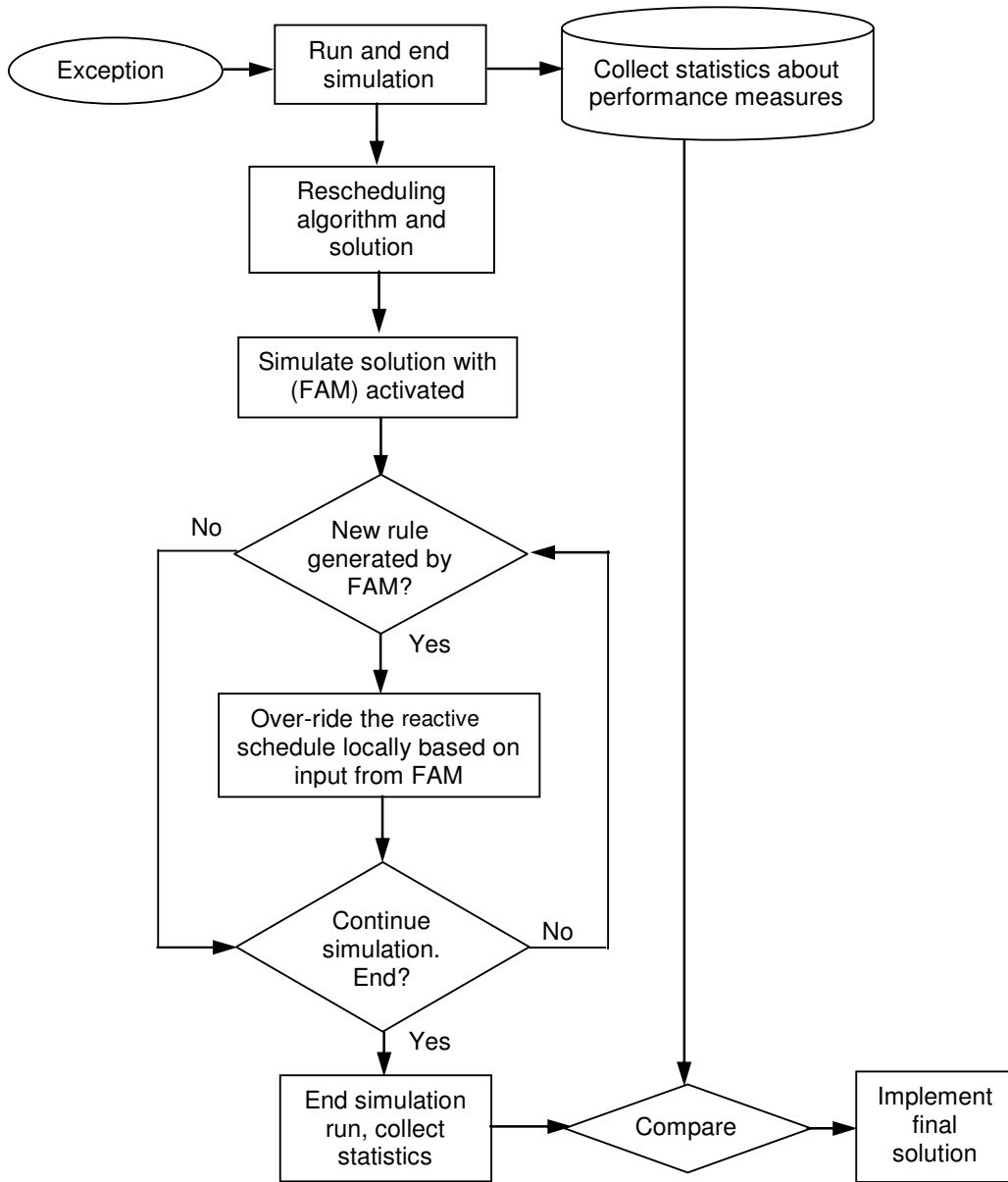
In the next section one more method of rescheduling is presented which focusses on how to reschedule as late as possible using sequence deviation as a tool to measure the results.

### **5.3.3 The selective re-routing approach for real-time control**

In this selective re-routing approach, a number of jobs are selected for re-scheduling with a view to make a local change to handle the exception in such a way that adaptation handling is carried out as late as possible in the planning



horizon. In the next section the overall concept is described, and then the detailed algorithm is explained with the help of an example.



**Figure 5.19: Post rescheduling analysis using simulation based FAM**

### 5.3.3.1 Concept of selective re-routing

Figure 5.20 shows the flow chart of the selective re-routing algorithm. As soon as an exception happens, an upper bound is computed using simulation. This

upper bound represents the worst case scenario as a result of not re-scheduling at all. A set of jobs is then created which are candidates for re-scheduling. These jobs are the ones which are placed beyond the time the exception occurred on the machine in the original (predictive schedule). Then, on the same stage where the exception happened, we determine the amount of capacity that is available on other machines using an algorithm. These jobs are then listed in a descending order (last job first, etc) from the order on the exception machine, and are listed into groups of jobs and iterations. So the first iteration has 1 job (the last job), the second iteration has two (the last and the second last). From this list, the first job is selected and re-scheduled on the machine with extra capacity on the exception stage. If there are multiple machines on the exception stage, the earliest machine available times on each machine would provide us with the information on where there is extra capacity. Since there are no more jobs in the 1<sup>st</sup> iteration, a simulation run is conducted to simulate this option, following which the results are stored in the database for analysis later. In the second iteration, there are two jobs (the last and second last) selected for rescheduling. From among these jobs, the second last job from the list is selected and rescheduled on the machine with the maximum capacity. Note that at the start of each iteration the same value on capacity is used initially. Following the rescheduling of the job, the capacity is updated again using the algorithm, and the last job is rescheduled at the machine with the next highest capacity on the exception stage. A simulation run is conducted at the end of the second iteration to compute exactly the result of performance measures (makespan, deviations) of rescheduling this specific constellation on the later stages, if any. Note that the second iteration will result in jobs rescheduled in the same sequence as they were in the predictive schedule. This is done to make sure, the makespan does not deviate too much due to sequence changes on the downstream stages. Note also that here, the job with the maximum processing time is not selected because the upperbound simulation considered in its calculation, the jobs in a particular sequence at the exception stage. Since when the calculations started upon the occurrence of the exception, the candidate jobs are then upstream in the system. Due to this, the phenomenon of having incorrect capacities used up by wrong jobs (jobs with lower processing times, appearing first in the sequence, and using up lower capacities) is less. Hence this same sequence is maintained during rescheduling in this method to make sure, the jobs appearing in a particular sequence use the first available higher capacity machines, than the higher processing time jobs who appear later in the sequence. Hence, the jobs from the descending order list, are used in the same sequence as explained.

This is followed by simulating these constellations and saving the results in the database. Depending on the number of iterations, this continues until all iterations have been considered. After considering jobs in the respective iterations

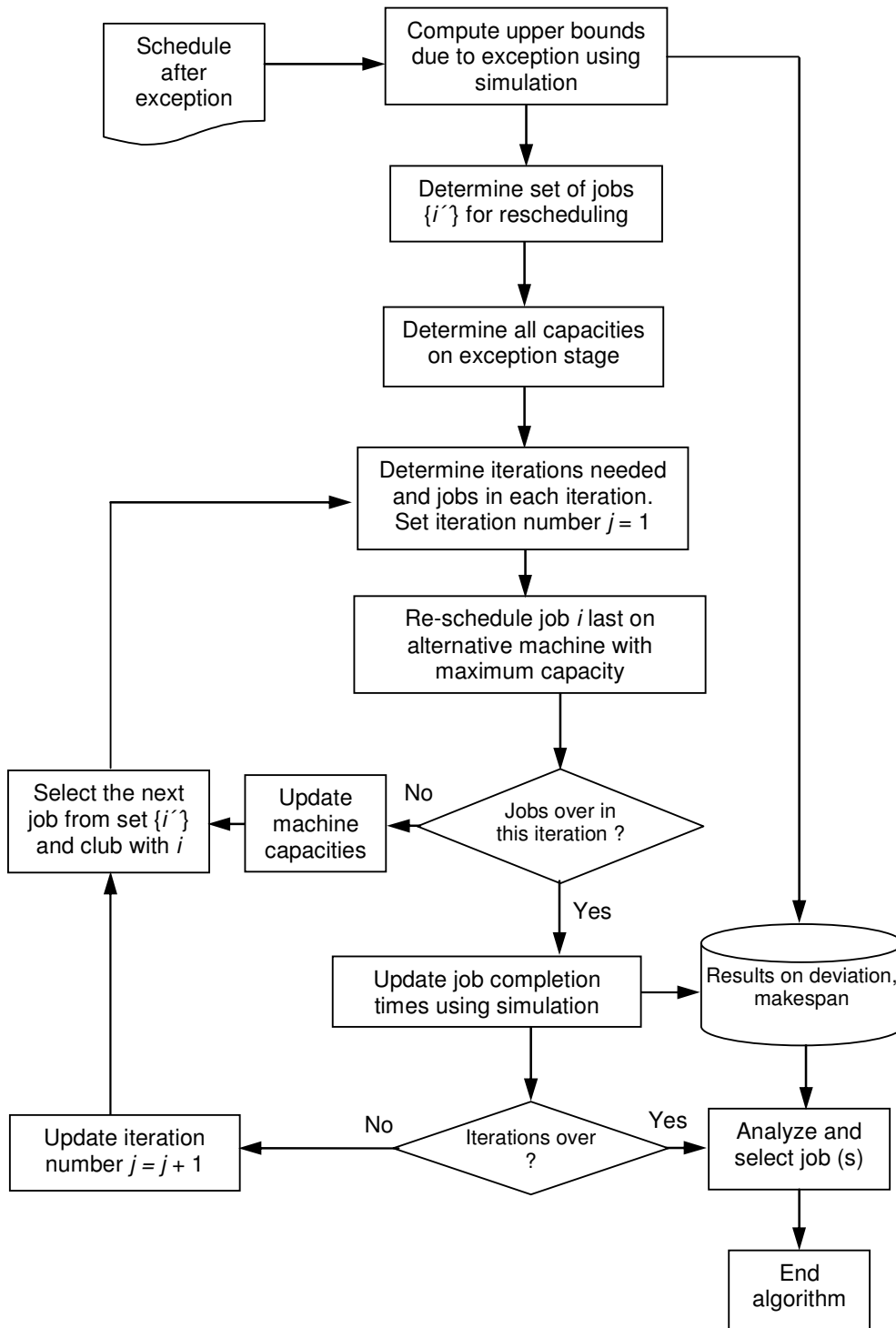


Figure 5.20 Selective re-routing for reactive scheduling

for re-scheduling, comparison is made between the upper bound and the result of re-scheduling the jobs. Depending on which performance measure the user chooses to be optimized, the respective solution is presented to the user. At this point, we would have computed the minimum number of jobs re-scheduled for a particular sequence deviation which is computed during the iterations, and we may also result in a better makespan depending upon the exception duration, process parameters like processing times, and capacity on alternative machines. Note that the reactive FAM is not used here because it is less likely that problems will arise due to changing the job sequences of the last jobs in the planning horizon. The detail algorithm for the selective re-routing is described in the next section.

### 5.3.3.2 Detailed algorithm for selective re-routing

The following are the steps in the selective re-routing algorithm, explained with the help of an example shown in Table 5.27. A 2x2 model was taken (2 machines on each of the 2 stages). The exception was set to occur from time 25 minutes, for a duration of 50 minutes on machine 2 on stage 1. The predictive schedule generated of this example is shown in Appendix 3, Figure A8 as a gantt chart. The exception would fall at the end of job 4 and continue until the end of job 5, shown by solid dotted braces.

**Table 5.27 Example to explain algorithm**

| <b>Jobs</b> | <b>Stage 1<br/>Processing times<br/>(minutes)</b> | <b>Stage 2<br/>Processing times<br/>(minutes)</b> |
|-------------|---|---|
| 1           | 25  | 20  |
| 2           | 25  | 20  |
| 3           | 25  | 25  |
| 4           | 30  | 20  |
| 5           | 15  | 15  |
| 6           | 30  | 20  |
| 7           | 10  | 20  |

1. Get the predictive schedule  $Sch_{predictive}$ . In this schedule all the jobs are scheduled.
2. Get time  $t_D$  = time when exception happened, and duration of exception as  $t_{Dur}$ . In this example is  $t_D = 25$  minutes and  $t_{Dur}$  is = 50 minutes.
3. Compute the upper bounds to compute the effect of the exception on the schedule, as the worst-case situation using simulation. At the end of the simulation run is the completely updated schedule which will result in the upper bound make-span value  $make-span^{UB}$ . Save this result in a database. In this

example, the simulation system calculates the upper bound value as 150 minutes. Note that the simulation also considers other details of the production system (one reason to use simulation for upper bound calculations).

4. Determine the number of jobs to reschedule as set  $\{i''\}$  which will include jobs on the machine  $k$  (where exception occurred), beyond the other earliest available machine  $a_{j,k}^m$  on stage  $j$  and the earliest available time for machine  $k$ , after computing the upper bounds. Create a list of all jobs in the set, in the order in which they enter the machine. In this current example, job 6 and job 5 are the candidate jobs, put in the set in this order.
5. Determine capacity on the exception stage by checking the earliest available time  $a_{j,k}^m$  for all the machines on this stage. The machine with the lowest earliest available time will have the highest capacity to absorb the exception, and will be the one where the job will be rescheduled to. Figure 5.17 shows the earliest available time  $a_{j,k}^m$ , on machines 2 and 3 for a generic example. In the current example, since there is only one machine on the exception stage,  $a_{j,k}^m = 85$  minutes (1:25:00.0000). This comes out as the time machine 1 on stage 1, becomes available earliest (can be calculated by adding up the processing times of all jobs on machine 1 on stage 1, and the idle times of the machine). Table 5.29 shows the capacities obtained for each iteration.
6. Determine the number of iteration as equal to the number of jobs in set  $\{i''\}$ . Classify each iteration with cumulative jobs from the set  $\{i''\}$ . For e.g. iteration 1 has job 1 from the set. Iteration 2 has job 1 and 2, iteration 3 has jobs 1, 2 and 3, and so on. In the current example, iteration 1 has job 5, and iteration 2 has job 6 and job 5. Note the sequence is the same in the second iteration as it was originally planned. Refer to Table 5.28 for the iteration and the jobs.
7. Set iteration number  $it = 1$ , or use the iteration number set by previous iterations.
8. Select the job  $i$  for this iteration and reschedule this job on the machine with maximum capacity. In the current example, job 5 is rescheduled on machine 1 on stage 1.
9. Run simulation with this change. Save results of makespan, starting time and sequence deviations in the database.
10. Update iteration number  $it = it + 1$ .
11. Select the jobs for this iteration. In this iteration job 6 and job 5 are selected.

**Table 5.28 Iterations and jobs selected for rescheduling**

| Iteration number | Jobs in iteration |
|------------------|-------------------|
|------------------|-------------------|

|   |              |
|---|--------------|
| 1 | Job 5        |
| 2 | Job 6, Job 5 |

**Table 5.29 Capacities on earliest available alternative machines**

| <b>Iteration</b> | <b>Earliest available machines time</b> |
|------------------|---|
| 1                | <b>85 minutes</b>                       |
| 2                |   |
| ...2.1           | <b>85 minutes</b>                       |
| ...2.2           | <b>115 minutes</b>                      |

12. Reschedule the first job on the machines with biggest capacities. In this example reschedule job 6 on machine 1 on stage 1 which has capacity of 85 minutes (since there was only one machine). Update capacity after this step, which then comes out as 115 minutes, or 85 minutes + 30 minutes. Reschedule the second job (in this case job 5 on machine 1 on stage 1).
13. Simulate this change and save results in the database as earlier.
14. If there exist more iterations, go to step 10 and repeat the steps, else depending on user settings, gather results from database and present to user.
15. End.

The same example is solved with the developed application and result is shown in Figure 9, A10 and A11. Table A65 and Table A66 show the detailed results. The results shown are for the second selected iteration in this example.

## **5.4 Conclusions**

In this research work, a simulation and optimization assisted scheduling and rescheduling system has been developed for a flexible production system configuration. The highlight of this system is the way simulation is combined with optimization. The optimization algorithm in the predictive phase uses the special property of the flow shop that the jobs flow in one direction only is taken advantage of in developing this procedure. The optimization algorithm considers other details like materials, tools and machine availabilities alongwith limited buffer capacities, alongwith demands on job delivery requirements and requirements of specific machines for some jobs, to compute a rough plan, which is further fine-tuned with the help of a rule-based simulation system. The simulation assisted rescheduling system also solves other issues not addressed by other research, in that it addresses the issues of when exactly to reschedule in the real production system, how to bring back the deviations to the planned trajectory as much as possible, how

to reschedule as less as possible, and if at all rescheduling is done, how can problems due to the rescheduling step be recognized and solved before the problems occur in the real world. Most importantly the rescheduling system also solves the problem of adaptation synchrony – the problem that in the real world changes take time which was not incorporated in computational systems developed by any researchers to date. Here too a rule-based simulation system is combined with an optimization algorithm which seeks to reduce the possible constellations available as solutions to the simulation system while also addressing the specific co-ordination and execution problems that occur due to rescheduling.

## Chapter 6 Overall framework and integration

### 6.1 Introduction

In this chapter we discuss an overall integrated framework for the scheduling and rescheduling of the production system configuration mentioned earlier. This framework is implemented in the next chapter. In this chapter, we also provide the building blocks of both the predictive and reactive systems on what each of them contain, and the capabilities of each block.

### 6.2 Overall system framework

Figure 6.1 shows the overall framework of our simulation assisted predictive-reactive approach for production scheduling and re-scheduling. In the predictive part, the planner can obtain one or more schedules which are generated with the help of the optimization algorithm and further fine tuned and analysed by the simulation system. In the reactive part, the execution is reacted by providing rescheduling solutions in real-time. The problem areas addressed at the beginning of the thesis are considered by this integration. In the next section, the building blocks used to perform the tasks shown in Figure 6.1 are described.

### 6.3 Integration of the entire system

Figure 6.2 shows the synthesis of the building blocks implemented as a black box using *Technomatix eM-Plant (SIMPLE++ first developed at the Fraunhofer IPA)*. These blocks are implemented as object libraries in *eM-Plant*. In the predictive system, the shifting bottleneck heuristic of Phadnis et. al. is used as a basic tool. This algorithm is extended further to include the delivery time and routing for standard as well as special jobs and other constraints for optimization. The simulation based FAM system with the rule generators are implemented as objects. The integration of simulation and optimization is achieved with the FAM system and these rule generators which control the simulation run execution.

The reactive system contains the objects of the rescheduling algorithms namely the match-up rescheduling and the selective re-routing algorithms. The Adaptation Synchrony Analysis algorithm extends the match-up rescheduling algorithm. The simulation based FAM object always uses the validity rule generator to avoid future problems due to the rescheduling step.



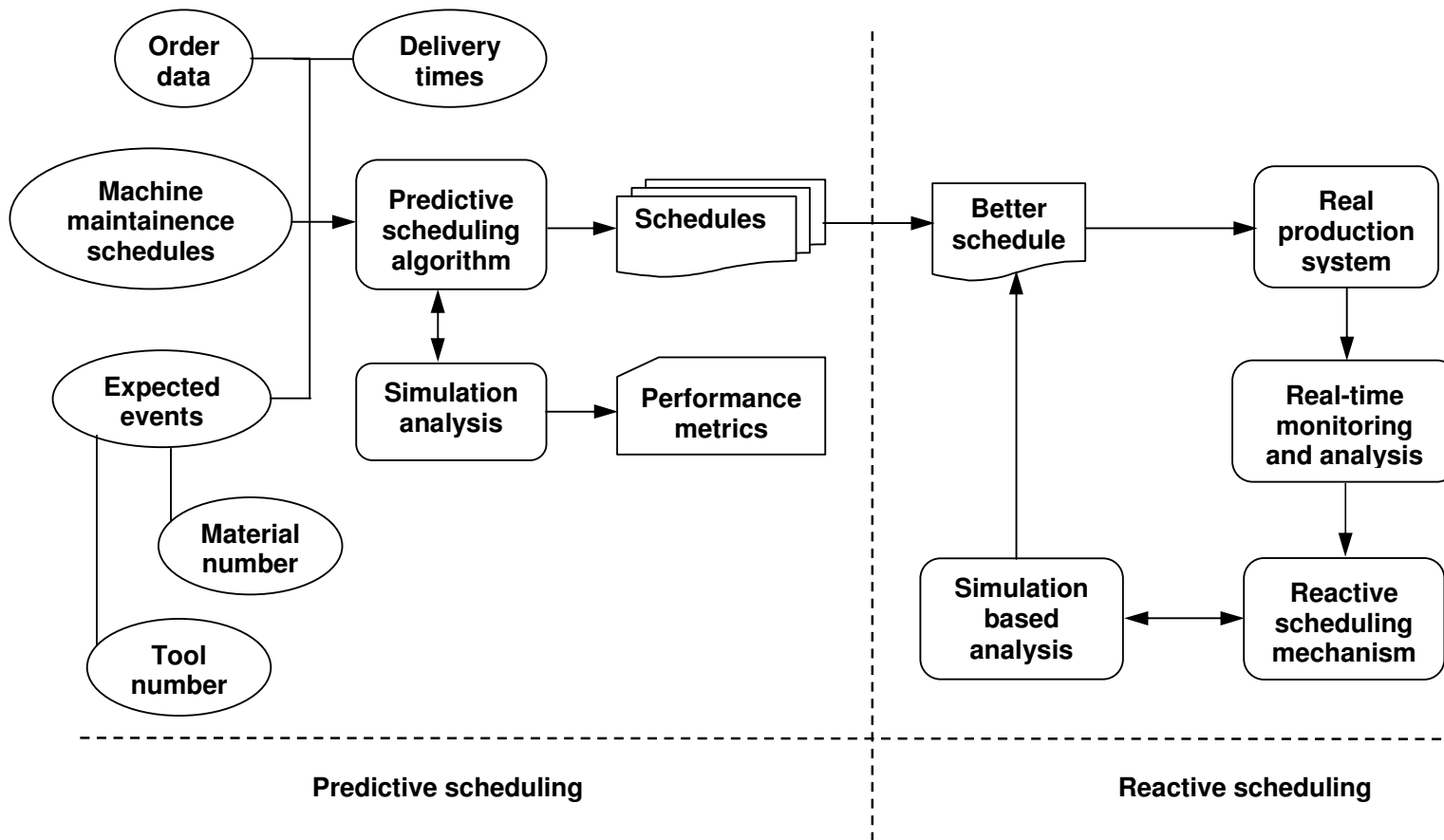
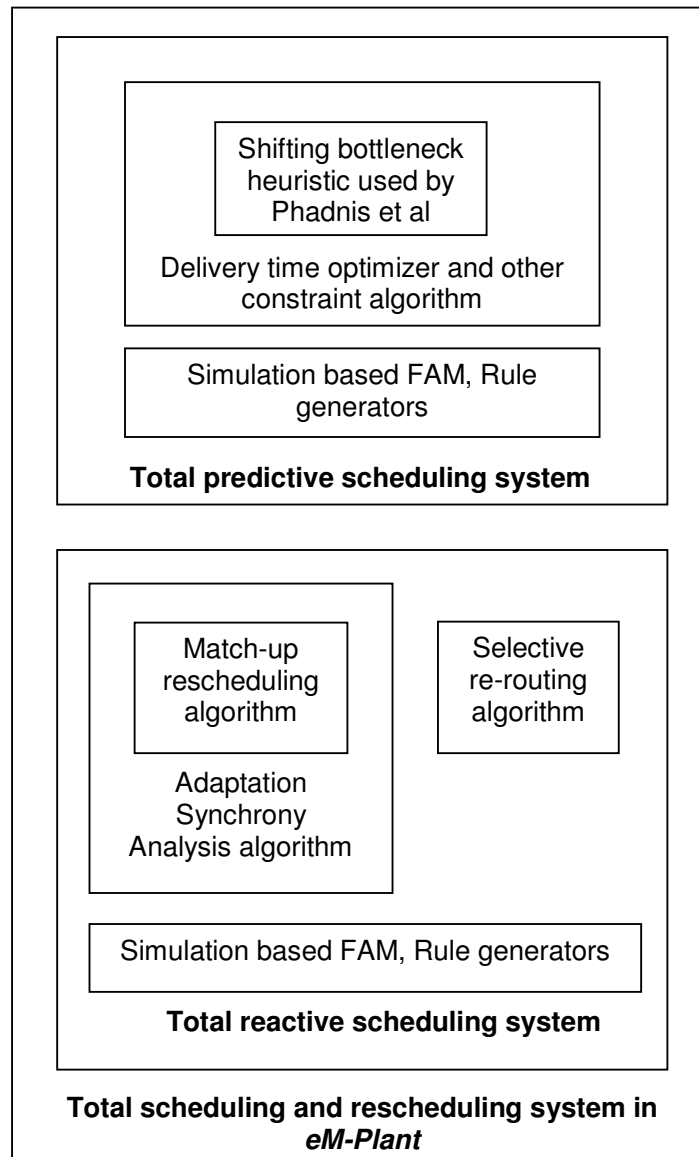


Figure 6.1: Overview of the simulation assisted production scheduling and re-scheduling system



**Figure 6.2 Building blocks of the total scheduling and rescheduling system**

## **6.4 Conclusions**

In this chapter, the blocks used for each of the systems developed in this thesis were described. In the next chapter, the implementation of these individual building blocks is explained alongwith details of each of them.

# Chapter 7    **Prototype software system realized**

## **7.1    Introduction**

In this chapter the predictive and reactive systems are implemented and details are discussed. Where required, necessary screen shots and the procedure to start-up and run the systems are described in details. This chapter starts with providing most important information about *eM-Plant* simulation software and its programming language SimTalk with necessary background information, to make the following sections more understandable. Then the structure of the developed system is described, followed by the application flow of the entire system, then followed by described in details the predictive and reactive systems.

## **7.2    Simulation software *eM-Plant***

As mentioned before, the simulation software *eM-Plant* is used as a platform for implementing the system. The software employs a graphical, object oriented approach for modelling and simulation per drag and drop. For this purpose it contains a library of generic objects that can be used to model a system. These objects can be placed per drag and drop into the basic modelling object called "Frames". Frames function as a container for simulation models. They encapsulate models and can be placed into other frames, thereby making the objects they contain reusable in bigger models. This method of placing frames with the functionality they provide into another frame is called hierarchical modelling.

Apart from frames, *eM-Plant* provides an object library that contains generic objects for modelling. These objects fall into four different categories, namely material flow objects, information flow objects, movable objects, and display and user interface objects. Material flow objects are used for modelling the physical outfit of the real-world system that is simulated. Information flow objects provide functions for storing and organizing data within the model, while the movable objects represent non-stationary entities within the model that can move through the material flow objects. Display and user interface objects handle the tasks of presenting data and interacting with the user. The most important objects from each of these categories that are also used within the system are described below:

1. Material flow objects

- a. Frame: As mentioned before, frames are the basic container for models in *eM-Plant*. All other objects are placed into them in order to form a simulation model. Frames can also be placed into other frames, thus reusing the model they contain in a bigger model.
- b. Connector: The connector is used for connecting two objects and establishing paths between them, that a movable object can be pushed along.
- c. Event controller: The event controller coordinates and synchronizes the events that occur during a simulation run. Since *eM-Plant* is a discrete event simulation system, events are inserted into the event list of the event controller at every time point where something happens within the simulation system that changes its state. One example is a movable objects entering a processing station. The station computed the time it takes to process the object. When this is done the event for the movable object to leave the processor is inserted into the event list at the correct time point. The event controller processes the different events in the event list sequentially, inserts new events that result from the analysed ones accordingly and advances the simulation time in the process.
- d. Source: The source is used for creating movable objects within the model, thus it is used as an entry place for movable objects into the model.
- e. Drain: The drain destroys movable objects that enter it. It is used as an exit from the system for the movable objects.
- f. SingleProc: The single processor is one of the different processor objects of *eM-Plant*. It is used to simulate some kind of processing station for the movable objects.
- g. Buffer: The buffer is used as a temporary storage for movable objects. They can simulate entry and exit buffers of machines, for example.
- h. FlowControl: The flow controller is used to diverge and converge the flow of materials in the system. It does not process the movable objects that pass through it, but distributes them among the objects that succeed the controller in the sequence of connected objects in the model.
- i. Track: The track is used to model paths for transporters within the system. The tracks are the only objects the transported objects can move on.

## 2. Information flow objects

- a. Method: The method object can store programming logic written in the language SimTalk. Programs stored in a method can be used for almost every task within the system and are therefore very useful and powerful.

- b. Variable: A variable is a small container that can be used to store a small amount of data, like a single integer, float or string.
- c. Table: Tables can be used to store and organize data in a matrix like data structure. It consist of several rows, columns, each with its own index, thus making every cell of the table clearly addressable and accessible.

### 3. Movable objects

- a. Entity: Entities are generic object for modelling anything within the model that is processed by the material flow objects. Entities can represent cars within a car factory, batches of circuit board, cogwheels, and wires. Virtually anything that is used in some kind of production process.
- b. Container: Containers are used for modelling all objects, that are used for transporting movable objects, but do not need to be processed and cannot move by themselves. Examples for real-world objects that they can represent are palettes or boxes.
- c. Transporter: Transporters simulate real-world objects that are used for transporting movable objects and can move on their own, but do not need to be processed. Examples for these objects are forklifts or automated guided vehicles.

### 4. Display and user interface objects

- a. Chart: Charts are used to present data in a graphical fashion. Normally this is data that *eM-Plant* collects during a simulation run.
- b. Dialog: Dialogs are used to communicate with the user of the simulation system. These objects provide text boxes, buttons, checkboxes, and all other elements that are common in modern computer dialogs.

#### 7.2.1 Programming language SimTalk

*eM-Plant* contains its own programming language called SimTalk. It can be used to implement custom logic and procedures within method object. The language is capable of changing the properties and behaviours of the different modelling objects, thus making it a very powerful tool. SimTalk features all the common control and data structures of model programming languages and can also use the methods that the different modelling objects provide. A SimTalk program is always structured in the following way:

```
[arguments]
[return value data type]
Is
```

```
[local variables]
Do
[program code]
end;
```

Within the arguments section, input variables can be defined that need to be handed over when the program is started. They can then be used within the program code. After this section the data type of the return value is defined, if the problem has one. The following *is* separates the former two sections from the area where the local variables are defined. These are only accessible within the program and are only existent as long as the program runs. The word *do* marks the start of the actual program, while *end*, marks the end of it. For more information about SimTalk see the webpage of Technomatix Corporation.

### 7.2.2 Important concepts of *eM-Plant*

While it is not possible and not the focus of this work to describe in detail the whole functionality that *eM-Plant* offers, some important features that were used during the implementation in either the predictive and the reactive part of the system are briefly described in the following:

1. Call back methods: *eM-Plant* offers the possibility for designing custom dialogs to interact with the user. The reactions to the user interactions with the dialog need to be implemented in a so called callback method. This method has to be divided into different sections. A callback argument can be assigned to all interactive elements of the dialog. When the user interacts with a certain element of the dialog, the callback method is called and the callback argument of the specific element is passed to it. The argument then determines which section of the method has to be executed.
2. Custom attributes: While all objects of *eM-Plant* have their standard attributes, it is possible to assign custom attributes to them. This is especially when a certain piece of data needs to be attached to an object. Within the system this is used for identifying the stage of the PMFS the object is located at or how many jobs are already waiting for capacity in a buffer.
3. Delivery tables: Moving objects are normally created within *eM-Plant* by a source following a random distribution. However, for the course of this work it was necessary to create moving objects in a previously determined sequence and at certain time points. For this purpose the concept of delivery tables was used. This table can be assigned to a source object, thus giving it exact instructions when to create how many of a certain type of

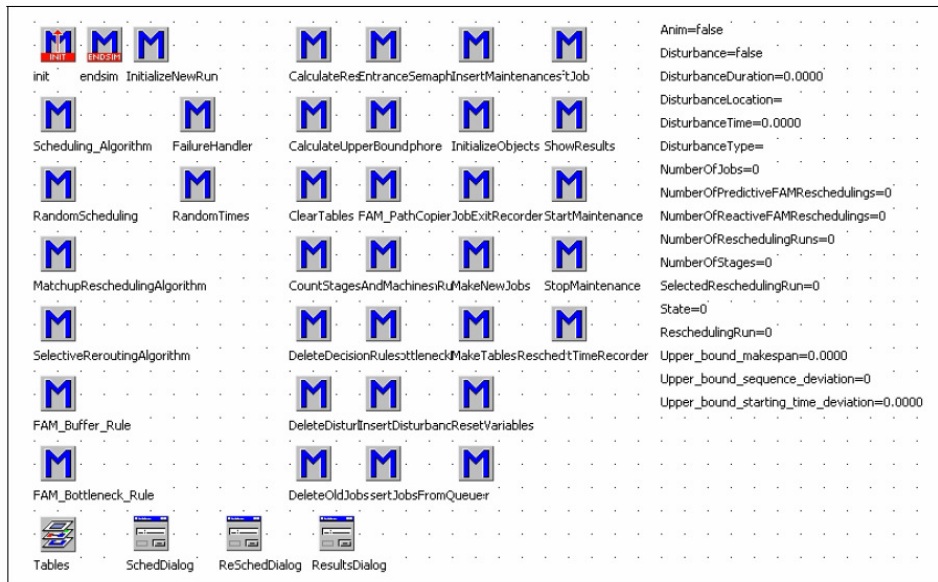
moving objects. Each row of the delivery table therefore contains a time point, a class of a movable object and the number of objects to be created.

4. Event triggered controls: Several material flow objects offer the possibility of executing a method upon the occurrence of certain events. This feature is used during the development of the system for the events of a movable object entering or leaving a material flow object and the event of a failure of a single proc object. The methods assigned to the event of a job entering or leaving a material flow object are denoted entry and exit control in the rest of the work.
5. *Init* and *endsim* methods: *eM-Plant* offers the possibility of automatically executing certain methods on the events of the initialization, the reset, and end of a simulation. In order for this to work, the methods must be named *init*, *reset*, *autoexec*, or *endsim*. When one of the before mentioned event occurs, *eM-Plant* automatically executes the corresponding methods. During the implementation of the system only the *init* and the *endsim* methods were used.
6. Processing time tables: While the processing times of the material flow objects normally follow some kind of random distribution, for the course of this work it was necessary to assign predefined processing times to the processors. For the purpose the possibility of assigning a processing time table to the processors was used. This table defines processing times for different classes of movable objects. Every time an object enters the processor the corresponding processing time of its class is selected from the table and used.
7. Suspending methods: *eM-Plant* offers the possibility to suspend the execution of a method and to wait for a particular event or a certain amount of time before the execution is continued. It is also possible to schedule a method call at some time point in the future. This is done by combining the call with a time value. The call is then executed after the defined amount of the time has elapsed.

### **7.3 Structure of the implementation**

The whole system is modelled into two frames, the top frame and the table frame. The table frame is inserted into the top frame. Therefore the user only needs to drag and drop the top frame into a PMFS model he wants to use for scheduling and rescheduling. A double click on the top frame of the system will not open the frame, but rather the predictive scheduling dialog, that is the user interface of the predictive part of the system. The whole analysis can be started from this dialog.

The top frame of the system itself contains several methods, variables and dialog objects that provide the functionality to solve the problems discussed in this thesis. The most important methods are located at the left of the frame, while all the secondary methods are located in the middle in alphabetical order. All variables can be found on the right side in alphabetical order, while the dialog objects are located at the bottom of the frame. A screenshot of the contents of the top frame is given in Figure 7.1.

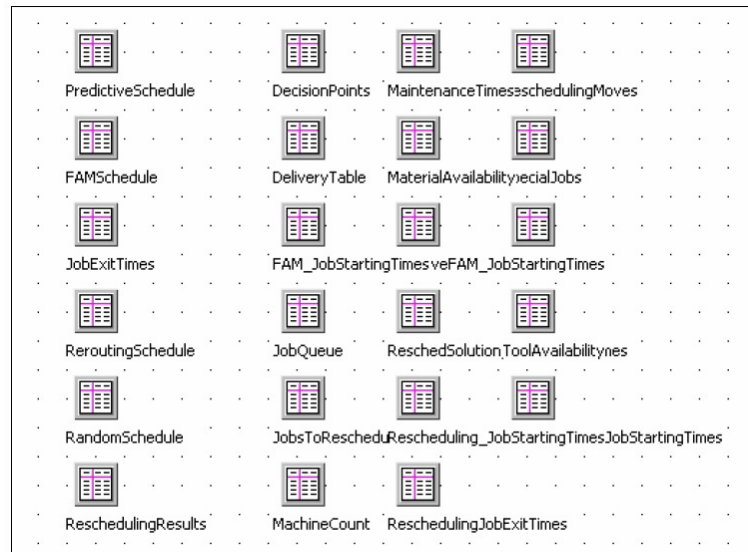


**Figure 7.1: Top frame of the developed system**

Right next to the dialog objects, the table frame can be found. This one is inserted into the top frame for structuring issues. It contains the tables that are used for organizing and storing the data of the developed application. A screenshot of the contents of the table frame is given in Figure 7.2. Note that the screen shot does not show all tables that are used during the scheduling process, since some are generated automatically by the system, based on the number of stages and machines the PMFS model contains.

The reader can refer to Appendix 4 for the description of all methods, tables and variables, described next, also described using diagrams and figures. The entry point of the whole system is the *InitializeNewRun* – method within the top frame. It is started after a double click on the frame containing the system. The method initializes the whole system for a new analysis and calls the *init* – method afterwards, which controls the application flow of the system with the *endsim* – method.



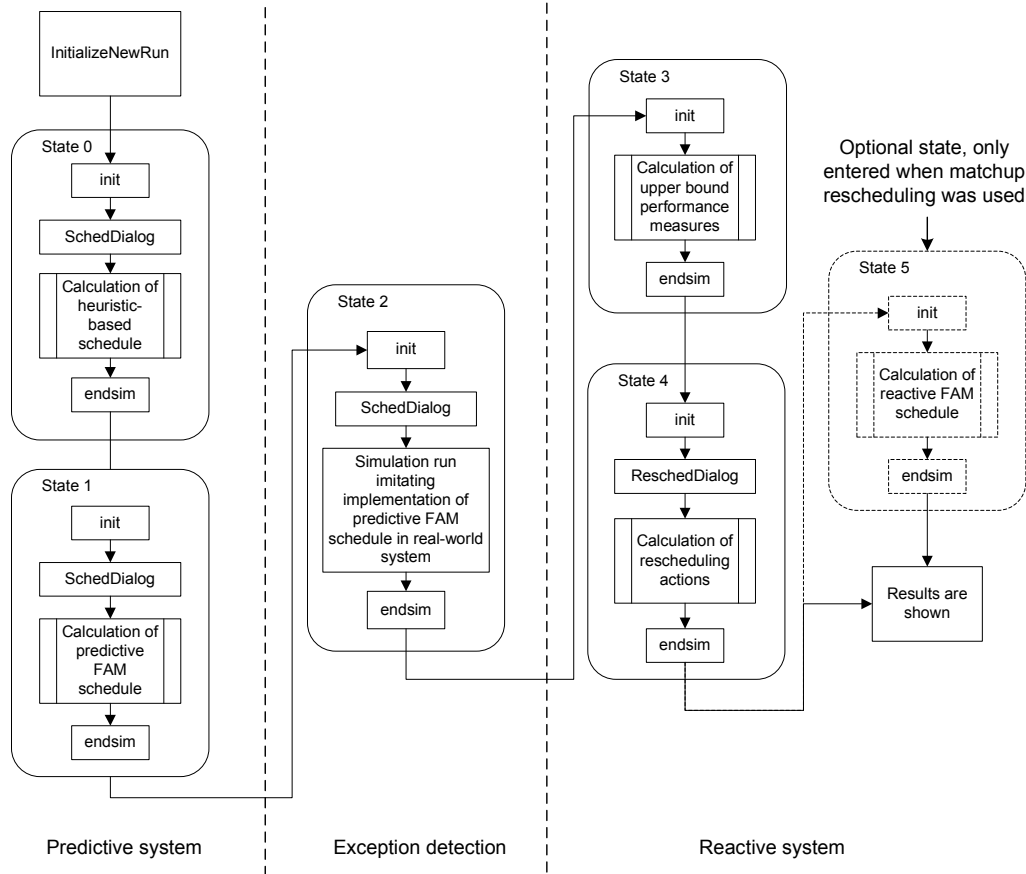


**Figure 7.2 Table frame which is part of top frame**

#### **7.4 Application flow of the implemented system**

The whole system is developed to function according to states. The different states define, at which point in the analysis the system is currently in and which data has already been calculated. The variable *State* keeps track of the current state of the system, while two special methods, called *init* and *Endsim* handle the transitions between the different states. The *init* – method is always executed right after the end of a simulation run. This makes these two methods perfect for controlling the behavior of the system, since the whole analysis typically consists of several simulation runs.

The application flow of the developed system can be seen in Figure 7.3. In the beginning the system is in state 0. A double click on the top frame of the system starts the *InitializeNewRun* – method which initializes the system for a new scheduling procedure by counting the stages of the PMFS model and the machines on each stage, creating new tables or deleting ones that are not needed anymore, adjusting the properties of the objects of the PMFS, deleting any data left from a previous analysis, and setting the variables of the system to their initial value. After this the *init* – method is called. This one evaluates the *State* – variable and opens the *SchedDialog* – Object, since the system is still in state 0 and the predictive FAM schedule has not yet been computed. The user has to generate the processing times and enter the number of jobs in this dialog, before he clicks the *OK* – button and starts the calculation of the heuristic – based schedule. The completion of this schedule will call the *Endsim* – method, which sets the state to 1, resets the



**Figure 7.3 Application flow of the system**

simulation software and calls the *init* – method again. This method evaluates the *State* – variable and opens the *SchedDialog* – object. Now the user can initiate the calculation of the predictive FAM schedule, by clicking the corresponding check box and the *OK* – button. After this calculation the *endsim* – method sets the state of the system to 2 and hands the control over to the *init* – method.

As soon as the predictive FAM scheduling is available, it is implemented in the real world production process. Its execution would then be monitored and the reactive part of the system would be started upon the occurrence of an exception. But since there was no real-world production system available during the course of this work, the simulation software was used to imitate the implementation of the predictive FAM schedule. Therefore in state 2 the *init* – method again opens the *SchedDialog* – object where the user gets a chance to activate the monitoring system, which checks for exceptions and activates the reactive system upon their occurrence. When an exception was found, the *endsim* – method sets the state of

the system to 3 and calls the *init* – method.

In state 3, the effects of the exception on the predictive FAM schedule are calculated, thus giving the upper bound performance measures for the reactive part. Also some other data is recorded, that is needed for the rescheduling algorithms. This step is executed automatically and sets the state of the system to 4 with the help of the *endsim* – method.

State 4 marks the beginning of the reactive part of the system. The *init* – method opens the *ReschedDialog* – object, where the user gets information about the detected exception. He can then select which rescheduling algorithm he wants to use to calculate the rescheduling actions. If he picks up the match – up rescheduling algorithm, he can also choose the criterion for selecting the best alternative from the different rescheduling possibilities and decide, whether he wants to analyse the selected alternatives with the reactive FAM. A click on the *OK* – button starts the process that calculates the different rescheduling alternatives and their performance measures. This is done automatically and generally involves several simulation runs. After the calculation of the last alternative the system presents the results of the rescheduling analysis.

If the user chose to analyse the rescheduling solution with the reactive FAM in the rescheduling dialog, the *endsim* – method sets the *State* – variable to 5. The *init* – method then automatically initiates another run of the simulation software using the selected rescheduling solution combined with the reactive FAM. In the next sections, the predictive and reactive systems are described.

## **7.5 Predictive scheduling system**

In the following sections, the prototype development for the predictive system is described. In the first section the issues with customization are presented followed by detailed description of the predictive system and its detailed components.

### **7.5.1 Predictive system**

#### **7.5.1.1 Customization of *eM-Plant***

The first step in the customization was to find a way of how to enter the needed input data for the algorithm in *eM-Plant* and how to save the results of the algorithm in such a manner, that a simulation run afterwards takes heed to these results. The problem of entering and saving the input data was solved by tables, which could be edited by the user. To do this in a more convenient way, a dialog was implemented which offered functionality to enter the needed data for the algorithm, namely the number of jobs, stages, machines on the different stages and

the processing times. In order to save the results and cause the simulation model to pay attention to these results, a combination of flow control units and attributes of the moving units was used. The flow control unit offers the functionality of routing MU's (Moving Units) according to the value of one of their attributes. Since the algorithm calculates a path for each job through the stages, which consists of the machines on each stage, this path can be written to each job as a set of attributes. This means that one attribute is needed for each stage, which should contain the number of the machine that processes the job at this stage.

Since the jobs enter the simulation system through the source in a sequential manner and therefore do not become available to the flow control unit simultaneously, a way had to be found to push the jobs in the system in the correct sequence. This means that the job that had to be routed first also had to enter the system first, otherwise the wrong job would have been routed to the machine and by this blocking it. This problem was overcome by the possibility of giving the source a delivery table, which specifies the type and order of the MU's it produces. This table could be filled by the algorithm thus generating the correct sequence.

After this concept was proven to be viable, the needed classes were derived from the basic classes of the *eM-Plant* class library and placed in a new folder called "Scheduling\_DSS". In the following section, the more specific aspects are described. **The reader is advised to look in Appendix 4 for description of each of the class objects (methods, tables and variables) developed, and their origin of derivation, to understand the descriptions.**

#### 7.5.1.2 Workflow of the whole system

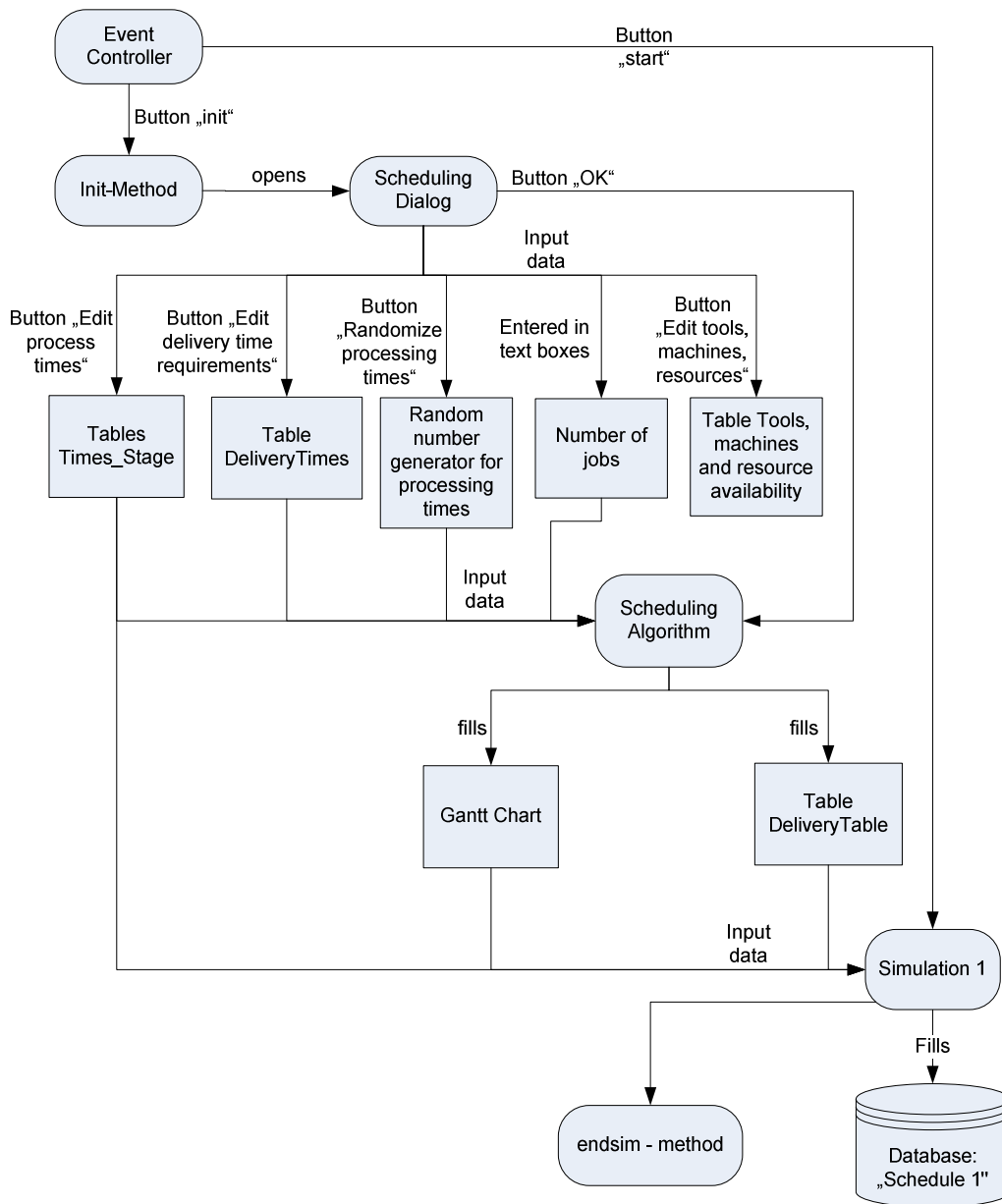
Refer to Figure 7.4 for an overall view of the system. Before the system can start, some initialization are done as follows. The *InitializeNewRun* – method is the entry point into the system. It is called as the user double clicks the top frame of the system. The method itself calls some other methods that set the initial data of the systems. First the *CountStagesAndMachines* – method is started, that counts the number of stages and the number of machines at each stage of the PMFS model the user has built and writes the values to the *NumberOfStages* – variable and the *MachineCount* – table. The method also creates some custom attributes within the objects of the PMFS model, which are needed for identification issues or during the process of shifting jobs between the machines. Next the *MakeTable* – method is called, which creates the needed amount of *JobSetInTime* – tables, *JobSetOutTime* – tables, *JobTransportationTimes* – tables, *Times\_Stages* – tables, and *Jobs\_Stage\_* – tables. Some of these tables are required for the rescheduling, which are nevertheless created here, are used in case exceptions happen later. One of the first four types of tables has to exist per stage, while of the last type is needed for each machine. The *InititalizeObjects* – method which is activated next, assigns the

delivery table to the source object of the PMFS model, the *JobExitRecorder* – method as exit control to the drain, and the *Router* – method as selection method to the flow controller objects. It also instructs the processor objects of the PMFS to use the *Times\_Stage* – table of their stage as source for their processing times and assigns them the *StartTimeRecorder* – method as entry control and the *FailureHandler* – method as failure control. After this the *ClearTables* – method and the *ResetVariable* – method delete any data, that might be left over from previous scheduling activities in the tables or variables of the system. Then the *RandomTimes* – method is executed which writes randomly generated processing times to the *Times\_Stage* – tables. Finally the control is handed over to the *init* – method via the event controller.

From this point on, the predictive scheduling dialog comes up. Refer Figure 7.4 for how the system works. The predictive scheduling dialog is the main interface between the user and the system. The dialog is used to collect all needed data and to start the predictive scheduling process. For this purpose it uses different text boxes, check boxes, and buttons. The dialog also contains a so called *callback* – method, which implements the actual functionality of the dialog. This method is a piece of code implemented in SimTalk that reacts to the interactions of the user with the dialog. A click on the button, for example, activates a part of the *callback* – method that carries out the actions, the button is supposed to start. Therefore interactive elements of the dialog can be marked with a *callback* – argument, which tells the *callback* - method which piece of code should be executed. The different elements of the dialog are shown in the coming section 7.5.3.1.

Using the dialog, the user enters the number of jobs, the processing times, the requirements on standard and special jobs with or without delivery constraints, and the various resources availability in the tables. The first step of the calculation of the heuristic – based algorithm to compute a predictive plan is started when the user clicks the *OK* – button of the scheduling dialog object (*SchedDialog*). The *callback* – method than starts the *Scheduling\_Algorithm* – method. The method gets its input data from the variables *NumberOfJobs*, *NumberOfStages*, and the tables *SpecialJobs*, *Times\_Stage*, *MachineCount*, *ToolAvailability*, *MaterialAvailability*, and *MaintenanceTimes*. Using this data it calculates a job sequence how the jobs should enter the PMFS and job routings for each job, i.e. on which machine the jobs should be processed on the different stages. The sequence is written to the table *DeliveryTable*, while the routings are stored in the table *HeuristicRoutings* and fills the standard data table of the standard Gantt chart object offered by *eM-Plant*.

After the *Scheduling\_Algorithm* – method has finished its calculations, a simulation run is started, which uses the data from the *DeliveryTable* – table for creating the jobs in the *source* – object of the PMFS model in the correct order. The



**Figure 7.4: Data Flow Chart of the predictive system**

flow controllers of the model route the jobs with the help of the *Router* – method according to the data stored in the *HeuristicRoutings* – table, while the processor objects gets the processing times from the *Times\_Stage* – tables. During the simulation run, the *JobExitRecorder* – method, which is assigned as entry control to

the sink – object of the model, records the time points, when the different jobs leave the system, in the *JobExitTables* – table, shown as “Schedule 1” in Figure 7.4. These times serve as a benchmark for the predictive FAM schedule, which is calculated in the next step. After the calculation is finished, the *endsim* – method is called, which sets the *State* – variable to 1, resets the simulation software and calls the *init* – method again.

## 7.5.2 Simulation assisted FAM system for predictive scheduling

### 7.5.2.1 Customization of *eM-Plant*

Several modifications have been done to customize *eM-Plant* for this problem. The needed classes were derived from the original ones and put into a folder called “Scheduling\_DSS”. Methods needed to be developed in place for controlling how the rule generators will be placed, and how the parts over-ride the original schedule of the optimization algorithm. The next section describes this in details. **The reader is advised to refer to Appendix 4 for a description of the methods, tables and variables discussed next.**

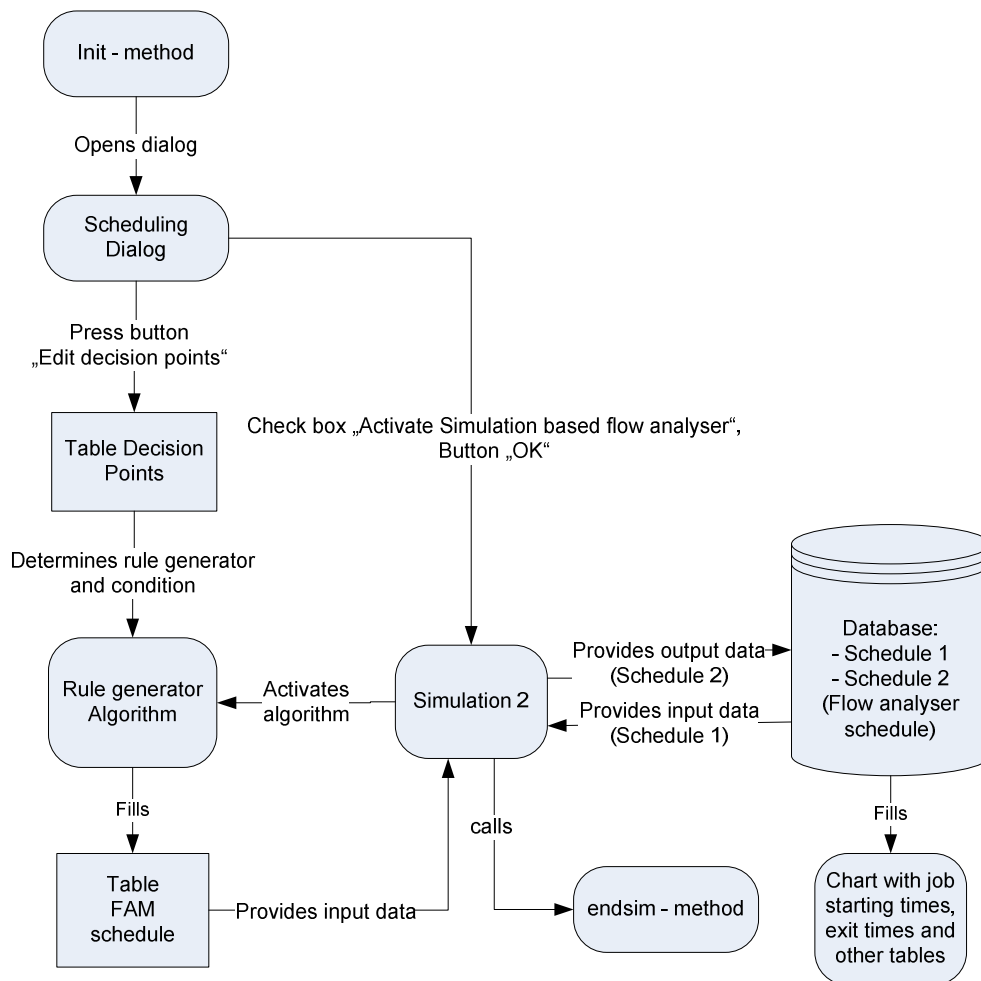
### 7.5.2.2 Workflow of the whole system

Figure 7.5 shows the workflow which is linked to the workflow of the predictive scheduling work flow of Figure 7.4 via the *init* - method. The second step of the predictive scheduling process is the analysis of the plan computed by the algorithm. It first opens the scheduling dialog, where the user can edit the decision points in the *DecisionPoints* – table and enter the rule generators at each decision point. Then the analysis is started when the user selects the check box *Activate simulation based flow analyser* and clicks the *OK* – button. Then the *callback* – method of the dialog calls the *InsertDecisionRules* – method which determines and inserts the rule generator methods namely the *FAM\_Bottleneck\_Rule* – method or the *FAM\_Buffer\_Rule* – method as entry controls into the decision points according to the decisions the user has entered in the *DecisionPoints* – table.

The *FAM\_PathCopier* – method is inserted into every decision point, where the user did not define a rule generator. This is needed, since the rule generators calculate the routings for the predictive FAM schedule. If a decision point would not contain a rule generator, no routings for the stage following the decision point would be generated. The *FAM\_PathCopier* – method handles this problem by simply copying the routings of the optimization – based plan to the predictive FAM schedule.

After this, another simulation run (shown as simulation 2 in Figure 7.5) is run with the job sequence from the *DeliveryTable* – table and the processing times from the *Times\_Stage* – tables is started. This time, the rule generators within the

decision points analyse the situation of the following stage and reroute the job if necessary according to the concepts of the FAM. These routings are stored in the *PredFAMRoutings* – table which is used by the *Router* – method in the flow controller objects in order to route the jobs correctly. The *StartTimeRecorder* – method, which is inserted as entry control in every processor object, records the job starting times and the sequence of jobs at every machine and stores them machine – wise in the *Jobs\_Stage\_Machine* – table as well as collected in the *FAM\_JobStartingTimes* – table. These tables are shown as the database where the schedule 2 is stored in Figure 7.5. Again, the *JobExitRecorder* – method saves the time points when the jobs exit the system in the *JobExitTimes* – table. As soon as all jobs are processed and the simulation run completed, the *endsim* – method is called, which sets the *State* – variable to 2 and starts the *init* – method.



**Figure 7.5 Data flow chart of the simulation assisted predictive FAM system**



At the end of the analysis the user can see both “Schedule 1” and “Schedule 2” in a graphical format using the *Gantt* – chart functionality of eM-Plant. This is done by filling up the chart with job finishing times for different scheduling methods. This way the schedules are stored and the user can compare different schedules.

### 7.5.3 Running the simulation and optimization based predictive scheduling system

This section describes how to start the predictive system in *eM-Plant* and run the simulation and the algorithms.

#### 7.5.3.1 Starting the system

After the model has been created, the system can be started. To get the initial state of the system the user has to do some tasks. At first he has to right click the *InitializeNewRun* – method. The system clears old table files from previous runs, if any and initializes other tables. After this the main starting dialog of the system comes up shown in Figure 7.6.

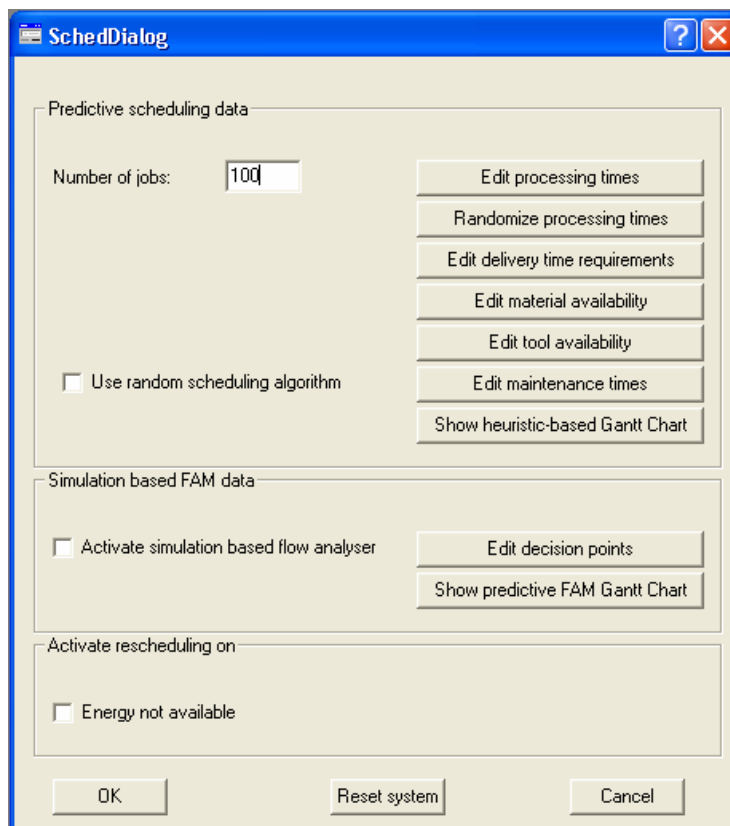


Figure 7.6: Screenshot of the scheduling dialog

Here the number of desired jobs need to be entered in the text box. Next the processing times are to be entered in the corresponding processing times table – for experimental purposes, this is done by clicking the *Randomize Processing Times* - button. The processing time table can be seen by clicking it where it contains the job names in the first column, while the processing times are in the second column. If some of the jobs are special jobs with or without fixed routings and with or without delivery time requirements within the factory, they are modelled by the user input by clicking the *Edit delivery time requirements* – button. A table will open up (Figure 7.8), with the names of the jobs in the first column. In order to mark a job as special, a path for it has to be entered in the second field of the according row.

|        | string 1 | time 2         |
|--------|----------|----------------|
| string | JobName  | ProcessingTime |
| 1      | Job1     | 3:20:00.0000   |
| 2      | Job2     | 3:40:00.0000   |
| 3      | Job3     | 2:55:00.0000   |
| 4      | Job4     | 3:40:00.0000   |
| 5      | Job5     | 45:00.0000     |

**Figure 7.7: Editing job processing times**

|        | string 1 | string 2        | time 3        | string 4 |
|--------|----------|-----------------|---------------|----------|
| string | JobName  | Path (4 Stages) | Delivery time |          |
| 1      | Job1     | 1,1,1,1         | 2:50:00.0000  |          |
| 2      | Job2     |                 |               |          |
| 3      | Job3     |                 | 1:50:00.0000  |          |
| 4      | Job4     |                 |               |          |
| 5      | Job5     |                 |               |          |
| 6      | Job6     |                 | 2:30:00.0000  |          |
| 7      | Job7     |                 |               |          |
| 8      | Job8     |                 |               |          |

**Figure 7.8 Editing job delivery time and routing constraints**

The path consists of integers, representing the number of the machine on each stage, separated by commas. Attention: The path has to consist of exactly as many integers as there are stages in the model. Both standard and special jobs have the possibility to set delivery dates as seen in Figure 7.8. The user can also enter in the material, resource and tool availability as seen in Figure 7.9 and 7.10. Then the user can edit the equipment availability plan to consider preventive machine maintenance programs as shown in Figure 7.11.

|        | string<br>1 | time<br>2    | string<br>3 |
|--------|-------------|--------------|-------------|
| string | JobName     | Available at |             |
| 1      | Job1        | 20:00.0000   |             |
| 2      | Job2        |              |             |
| 3      | Job3        | 40:00.0000   |             |
| 4      | Job4        |              |             |
| 5      | Job5        | 1:30:00.0000 |             |
| 6      | Job6        |              |             |
| 7      | Job7        |              |             |
| 8      | Job8        |              |             |

**Figure 7.9 Editing tool or resource availability**

|        | string<br>1 | time<br>2    | string<br>3 |
|--------|-------------|--------------|-------------|
| string | JobName     | Available at |             |
| 1      | Job1        | 20:00.0000   |             |
| 2      | Job2        |              |             |
| 3      | Job3        | 40:00.0000   |             |
| 4      | Job4        |              |             |
| 5      | Job5        | 1:30:00.0000 |             |
| 6      | Job6        |              |             |
| 7      | Job7        |              |             |
| 8      | Job8        |              |             |

**Figure 7.10 Editing material availability**

After all this data has been entered the *OK* - button can be clicked shown in Figure 7.6, which will start the scheduling optimization algorithm and the predictive simulation run as shown and explained in previous sections. After the end of the simulation the dialog (Figure 7.6) re-appears where the user can click the *Activate simulation based flow analyser* - checkbox activated. Then the user enters the data for the simulation based flow analyser. By pressing the *Edit decision points* - button the corresponding table (Figure 7.12) will show up and the user can select the rule generators and conditions for each decision point.

|        | string<br>0     | time<br>1         | time<br>2       | string<br>3 |
|--------|-----------------|-------------------|-----------------|-------------|
| string | Machine name    | Maintenance start | Maintenance end |             |
| 1      | Machine1_Stage1 |                   |                 |             |
| 2      | Machine2_Stage1 |                   |                 |             |
| 3      | Machine3_Stage1 | 1:30:00.0000      | 2:00:00.0000    |             |
| 4      | Machine4_Stage1 |                   |                 |             |
| 5      | Machine5_Stage1 |                   |                 |             |
| 6      | Machine1_Stage2 |                   |                 |             |
| 7      | Machine2_Stage2 | 2:00:00.0000      | 2:30:00.0000    |             |
| 8      | Machine3_Stage2 |                   |                 |             |
| 9      | Machine4_Stage2 |                   |                 |             |
| 10     | Machine5_Stage2 |                   |                 |             |
|        | Machine1_Stage3 |                   |                 |             |

**Figure 7.11 Editing equipment availability**

Once again, the user clicks the *OK* – button, and now the user can start the simulation again with the flow analyser activated. The scheduling dialog once again disappears as the simulation runs and re-appears in the end, where the user can select the *Use random scheduling algorithm* - checkbox and de-select the *Activate simulation based flow analyser* - checkbox deactivated. This calculates a plan randomly and runs the plan to calculate a schedule. The simulation run with the random schedule doesn't need any configuration. The user has just to click the *OK* - button one more time. After these three simulation runs (one with the schedule from optimization algorithm, one with the flow analyser activated and one with a random schedule) the user can analyse and select a schedule he likes to implement in the real world. At appropriate times, the user can compare the *Gantt* – chart objects to display the schedules, or alternatively, to get the job finishing

times and the job finishing sequence the user needs to open the *JobExitTimes* - table.

|        | string 1             | string 2            | string 3   |
|--------|----------------------|---------------------|------------|
| string | Decision point       | Rule generator      | Condition  |
| 1      | DecisionPoint_Stage1 | FAM_Buffer_Rule     | Optimality |
| 2      | DecisionPoint_Stage2 | FAM_Buffer_Rule     | Optimality |
| 3      | DecisionPoint_Stage3 | FAM_Bottleneck_Rule | Validity   |
| 4      | DecisionPoint_Stage4 | FAM_Buffer_Rule     | Optimality |

**Figure 7.12** Editing the decision points, conditions and rule generators

## 7.6 Reactive scheduling system

In the following sections, two separate methods and their implementation issues and components are discussed.

### 7.6.1 Match-up rescheduling system

In the following sections the match-up rescheduling system is described. The first section gives an idea of issues existing to develop such a system within *eM-Plant*. Then the detailed workflow of the system is described which mentions how the system was actually implemented and integrated with *eM-Plant*.

#### 7.6.1.1 Customization of eM-Plant

Several modifications have been done to customize *eM-Plant* for this problem. Methods and tables had to be implemented for managing the data resulting from the algorithms and simulation runs. The result of the simulation runs needed to be recorded into tables which functions like a database to store the job finishing times, job starting time deviations, job sequence deviations and the make-span.

To control the workflow of the reactive system and for the user to select appropriate actions for rescheduling, a graphical user interface has been developed which allows the user to run the simulations and algorithms for the type of methods selected. To make sure the rescheduling does not affect future performance,

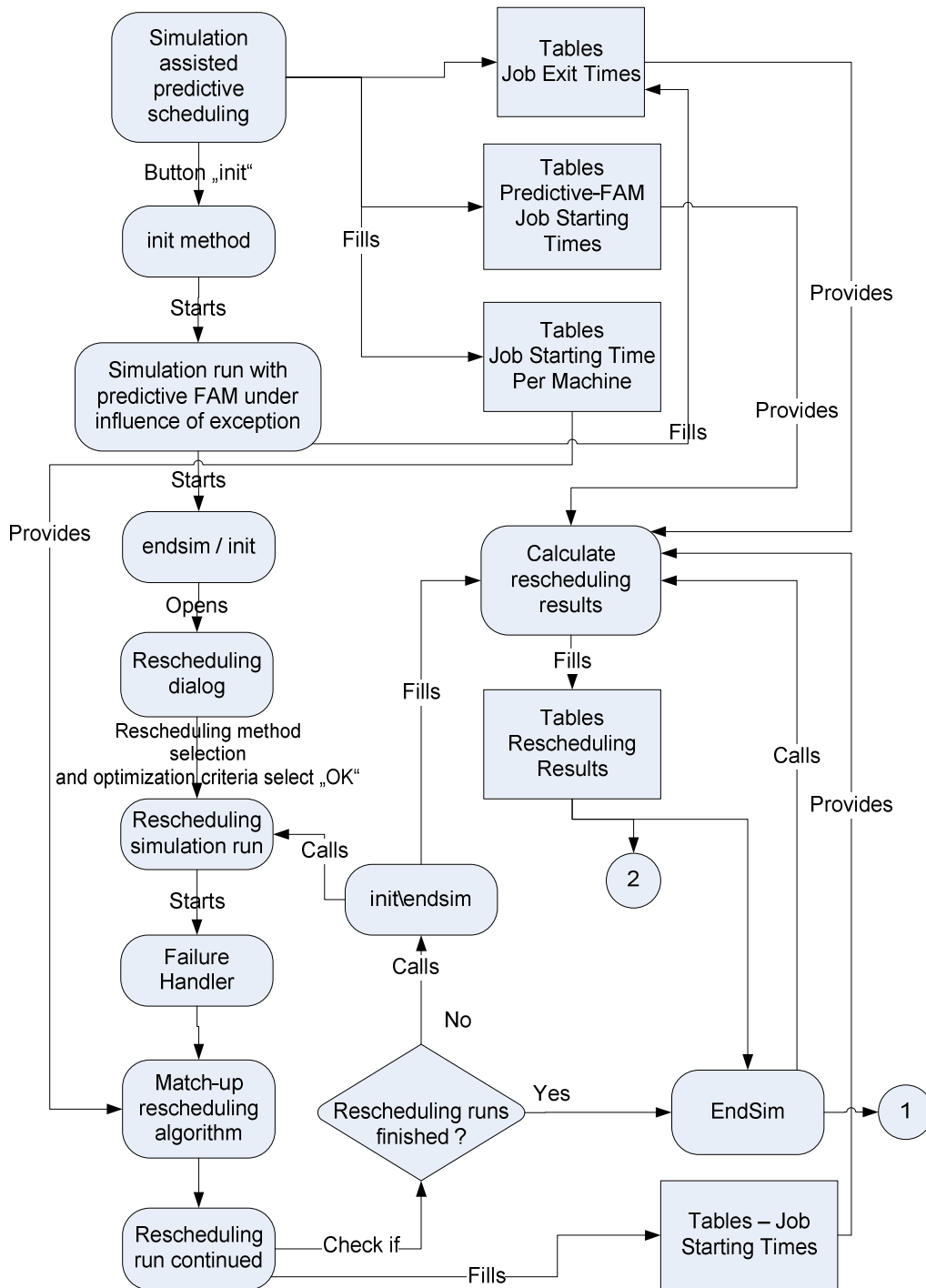
methods have been implemented for inserting automatically validity rule generator into the decision points at appropriate points in the simulation model. The customization takes its origin from the basic classes of the class library of *eM-Plant*. The needed classes were derived from the original ones and put into a folder called "Scheduling\_DSS". **The reader is advised to refer to Appendix 4 for the description of the methods, tables and variable objects, which are described to develop this system.**

### 7.6.1.2 Workflow of the system

Figure 7.13 shows the overall workflow of the simulation assisted match-up rescheduling system. Normally the predictive FAM schedule would be implemented in a real-world production system. The execution of the schedule would be monitored by a real-time monitoring and control module that would detect any exceptions that arise. Since no real-world production system is used during the course of this work, the simulation software was used for implementing the predictive FAM schedule and detecting exceptions.

For this purpose a feature of *eM-Plant* was used, that automatically activates a method when a failure in one of the processing objects arises. The functionality to handle an exception is implemented within the *FailureHandler* – method. This method saves information about the exception in the *DisturbanceDuration* – variable, the *DisturbanceLocation* – variable to true, in order to signal all other components of the system, that an exception was detected. The next time the *endsim* - method is called it evaluates the *Disturbance*-variable and sets the state of the system to 3, which initiates with a call of the *init* - method the calculation of the upper bound measures. Then the calculation of the upper bound performance measures is done by calculating the impact of the disturbance on the performance of the predictive FAM schedule. Therefore another run of the simulation software is automatically initiated as seen in Figure 7.13, shown after the *init* - method. The job sequence and the processing times remain unchanged and the job routings are determined by the *PredFAMRoutings* - table.

During the simulation run the *StartTimeRecorder* - method records the time points, when the jobs start processing on the different machines and records them machine-wise in the *Jobs\_Stage\_Machine* - tables as well as collected in the *Upper\_bound\_JobStartingTimes* - table. After the run has finished, the *endsim* - method is activated which sets the state of the system to 4 and calls the *CalculateUpperBound* - method. This method uses the values from the *FAM\_JobStartingTimes* - table and the *Upper\_bound\_JobStartingTimes* - table to calculate the starting time deviation the exception caused. This value is stored in the *Upper\_bound\_starting\_time\_deviation* - variable. The method also stores the time the predictive FAM schedule needs to be executed under the influence of the



**Figure 7.13 Data flow chart of the optimization and simulation based match-up rescheduling system**

disturbance in the *Upper\_bound\_lead\_time* - variable. These values serve as benchmarks for the following rescheduling actions, because they represent the option of not reacting to the exception at all. After this the *init* - method takes over again as seen in Figure 7.13, and opens the *ReSchedDialog* - object.

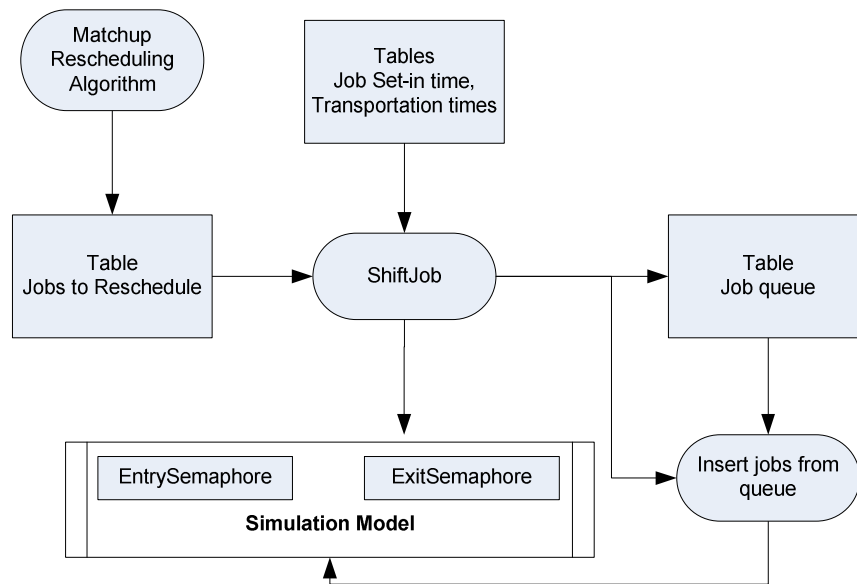
The *ReSchedDialog* - object serves as the main interface between user and system during the reactive scheduling phase. Within this dialog the user can choose the rescheduling method (algorithm) he wants to use (in this case, he uses the match-up method) in order to react to the detected exception. For this purpose the dialog contains text fields showing the gathered information about the exception and check boxes for selecting the rescheduling algorithm. Like the *SchedDialog* - object this dialog also contains a *callback* - method that implements the functionality of the dialog elements.

When the match-up rescheduling algorithm is selected and an *OK* – button is clicked, the system directly starts a simulation run with the job sequence from the *DeliveryTable* - table and the routings from the *PredFAMRoutings* - table. The simulation run is executed up to the point where the disturbance was detected. Then the *FailureHandler* - method calls, as seen in Figure 7.13, the *MatchupReschedulingAlgorithm* - method which implements the functionality desired. The method uses information from the *DisturbanceLocation* - variable, the *Jobs\_Stage\_Machines* - tables and the *Times\_Stage* - tables to generate the set of candidate jobs for rescheduling and write in the correct order into the *JobsForRescheduling* - table. The number of jobs in this set also gives the number of rescheduling alternatives which will be calculated in the following way. This value is stored in the *NumberOfReschedulingRuns* - variable, which serves as a control variable to keep track of the already calculated alternatives. Finally the *MatchupReschedulingAlgorithm* - method schedules a job shift for every job that needs to be relocated in the current rescheduling alternative and writes the shifts to the *ReschedulingMoves* - table for alter use in the reactive FAM. The job shifts are scheduled by performing a delayed method call of the *ShiftJob* - method for every job that needs to be shifted. The data in the *JobSetOutTimes\_Stage* - tables determines how long the method calls have to be delayed. This is part of the ASA (Adaptation Synchrony Analysis) system described in chapter 6, and is shown in Figure 7.15, how it interacts with the match-up rescheduling algorithm. The delay simulates the time needed to take a job out of its current buffer. The exact job shift procedure is described in the next paragraph, which incorporates the concept of the ASA.

During the rescheduling process with the match-up rescheduling algorithm jobs need to be shifted from the buffer of one machine to another. The *ShiftJob* - method handles this task, and a schematic view of the entire process is shown in Figure 7.14. It is called by the *MatchupReschedulingAlgorithm* - method with a



delayed method call. This delay represents the time needed to take a job out of its current buffer. The first action of the *ShiftJob* - method is to delete the job from its current buffer. After this it waits for an amount of time, which represents the time needed to transport the job from its original buffer to the new one. The method gets this time value from the according *JobTransportationTimes\_Stage* - table. When this time has elapsed, the job has virtually arrived at its new buffer and the process to insert it into its new destination can start.

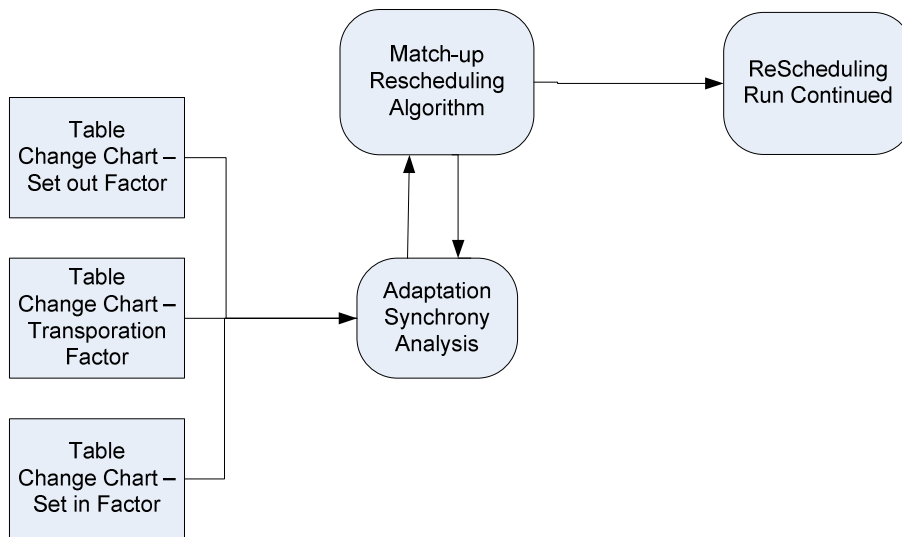


**Figure 7.14 Detailed working of the job shifting procedure**

Therefore the *ShiftJob* - method checks whether the new destination buffer has enough capacity for the new job. This is the case if the remaining capacity of the buffer minus the number of currently running insertion processes of other jobs is at least 1. Note that more than one insertion process can be executed for a particular buffer simultaneously. The number of currently running insertion process is stored in a custom attribute of the buffer. If there is enough capacity, the number of currently executed insertions is increased by 1 and the *EntrySemaphore* - method and *ExitSemaphore* - method are inserted as entry and exit control respectively into the destination buffer. The reason for this is, that the insertion of the job takes an amount of time, specified in the corresponding *JobSetInTimes\_Stage* - table, for which the execution of the *JobShift* - method is paused. During this time it has to be ensured that the buffer has a remaining capacity that is equal to the currently running insertions, in order for the insertion processes to work. The *EntrySemaphore* - method and the *ExitSemaphore* -

method handle this task, by only letting jobs enter the buffer if its remaining capacity is greater than the amount of running insertion processes. Otherwise they refuse jobs to enter the buffer. After the time needed for the insertion process has elapsed, the *ShiftJob* - method is reactivated. It inserts the job into the buffer, decreases the number of currently executed insertions, and deletes the entry and exit controls if no more insertions are currently executed at this buffer.

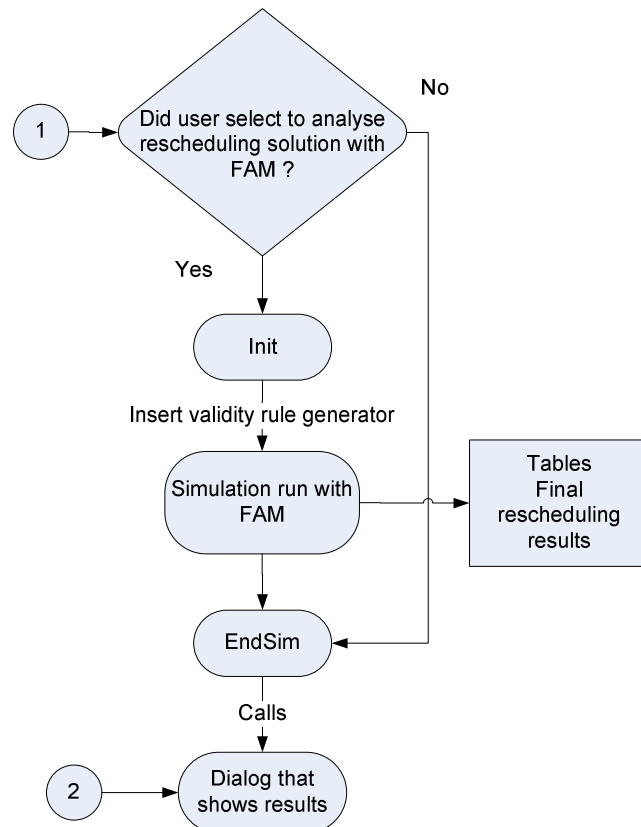
If the *ShiftJob* - method finds that the buffer does not have enough capacity after the transportation time of the job has elapsed, the job is written to the *JobQueue* - table, which serves as a virtual queue. Then the *InsertJobsFromQueue* - method is inserted as exit control for the destination buffer. The *ShiftJob* - method terminates and leaves the rest of the task of inserting the job into the buffer to the just inserted exit control. The *InsertJobsFromQueue* - method waits for a job leaving the buffer, which should cause the buffer to have enough capacity for the job in the queue. The method then starts the normal insertion process described above and removes itself as exit control from the buffer, if no more jobs are waiting to be inserted into it. After the job shifts are calculated and initiated the simulation run continues. Again the *JobStarttimeRecorder* - method records and saves the time points the jobs start processing on the machines in the *Rescheduling-JobStartingTimes* - table, while the *JobExitRecorder* - method saves the time points the jobs leave the system in the *ReschedulingJobExitTimes* - table.



**Figure 7.15 Interaction of the ASA module with the rescheduling system**

After the simulation run is finished the *CalculateReschedulingResults* - method is called as seen in Figure 7.13, which computes the lead time, sequence

deviations, and starting time deviation of the current rescheduling alternative using the *Jobs\_Stage\_Machine* - tables, the *FAM\_JobStartingTimes* - table, and the *ReschedulingJobStartingTimes* - table. The results are stored in the *Rescheduling* - table.



**Figure 7.16 Continuation of the simulation to conduct reactive FAM analysis**

As seen in Figure 7.13, finally the *endsim* - method is started which checks whether all rescheduling alternatives have already been calculated. If not, the whole process is started anew by calling the *init* - method. Otherwise the *ShowResults*-method is called, which opens the *ResultsDialog* - object, that presents the rescheduling results to the user using the *ReschedulingResults* - table, thus marking the end of the whole rescheduling process. After the simulation ends, and the rescheduling solution has been calculated, the system may continue further if the user selected to continue the reactive FAM analysis to solve future problems due to rescheduling, shown by rounded number 1 and 2 in Figure 7.13. Figure 7.16, shows the continuation. If yes, then the simulation model is initialized with the validity rule generator and inserted in the appropriate decision points, and the

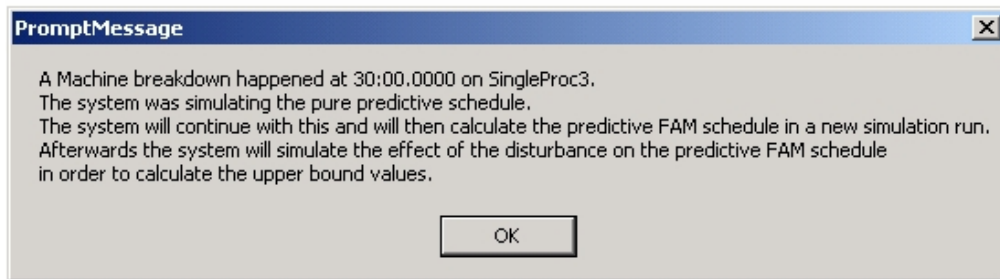
simulation starts once again. The reactive FAM works very similar to the predictive one. However it only uses the *FAM\_Bottleneck\_Rule* - rule generator method as entry controls within the decision points. Also the analysis only starts after the occurrence of the exception, since before that time point the original predictive FAM schedule is executed, which has already been analysed by the predictive FAM.

The simulation software therefore executes the routings of the *PredFAMRoutings* - table up to the point, where the exception occurred. Then the *FailureHandler* - method is called, which uses the *SelectedReschedulingRun* - variable and the *ReschedulingMoves* - table in order to look up the selected rescheduling solution and the job shifts that were executed in it. It then schedules a *ShiftJob* - method for every job shift that needs to be executed and also calls the *InsertBottleneckRuleForFAMResched* - method. This method inserts the *FAM\_Bottleneck\_Rule* - method as entry controls within the decision points. The job shifts are executed as described in the earlier sections. Of course the *JobExitRecorder* - method and the *StartTimeRecorder* - method record the time points when the jobs leave the system or start processing on the machines and store them within the *ReschedulingJobExitTimes* - table or the *ReactiveFAM\_JobStartingTimes* respectively. The analysis ends with a call of the *CalculateReschedulingResults* - methods which calculates the lead time, the sequence deviation, and the starting time deviation compared to the performance measures of the predictive FAM schedule, using the *Jobs\_Stage\_Machine* - tables, the *ReactiveFAM\_JobStartingTimes* - table, and the *FAM\_JobStartingTimes* - table. Finally the results are presented to the user. In the end of the simulation run, the results are presented to the user, using data from the corresponding table files.

### 7.6.1.3 Running the simulation and algorithms

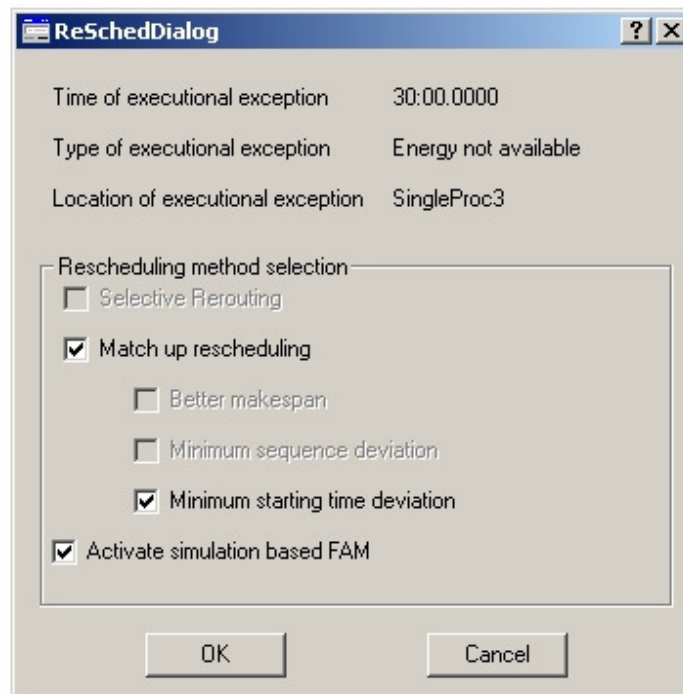
The user first right clicks the *InitialiseNewRun* – method which opens up the predictive system dialog as explained earlier. This is done to collect data on performance indices of the predictive run – this data is used later to generate results of the rescheduling system. In order to activate the rescheduling system, he has to activate the check boxes *Energy not available* to manage a generic exception, in the group box “Activate rescheduling on”. On clicking the *OK* - button, the predictive algorithm runs and the simulation starts. This simulation run is the predictive simulation until we have the predictive FAM schedule. As soon as the exception happens, the user is notified of the exception, and the information about the exception like time of exception and location of exception. This is shown in Figure 7.17.

Note this is done to identify which system the user is in – the predictive system or the reactive system. So, for example, if no exception is inserted, then the system knows it is in the predictive phase, and vice versa, so that that the



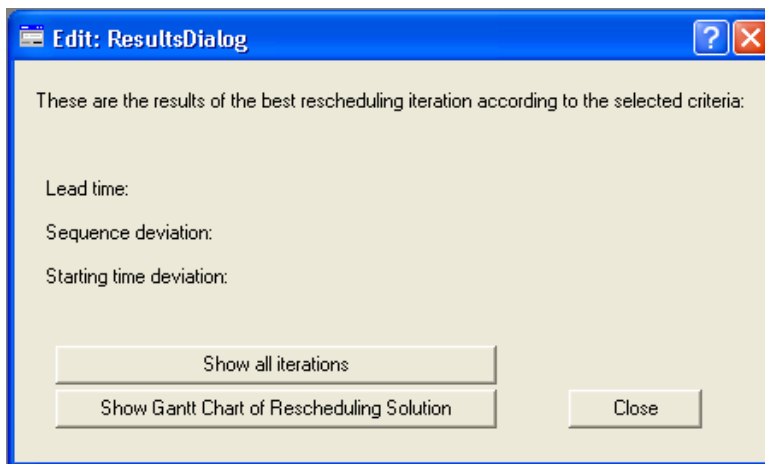
**Figure 7.17 Screen shot for user notification of exception**

developed application can take control of the future actions. On clicking the *OK* - button, the predictive simulation runs. Next, the rules for the decision points are inserted as the user has inserted them in the decision points table. This will result in the predictive FAM schedule. After this the predictive FAM with the exception simulation run will start and end which will compute the upper bounds. In the next simulation run, the rescheduling run is simulated. In this run, the user is presented with the rescheduling dialog as seen in Figure 7.18, where he can select his optimization options and criteria. Options are *Match up rescheduling*, with sub-opti-



**Figure 7.18 Screenshot rescheduling dialog**

ons with only one selection between *Better make-span*, *Better starting time deviations* or *Better sequence deviations*, and similarly for other options. At this point, the user can also select the check box *Activate simulation based FAM* for analysing the rescheduling solution is he wishes so. As soon as the user clicks the *OK* - button, the match-up rescheduling algorithm (or the respective algorithm) computes the solutions and the simulation iterations will start. Within these iterations, the system will compute the performance indices the user selected, and run the simulation once again with the FAM activated. At the end of the simulation the results will be presented to the user in the form of a pop-up window as seen in Figure 7.19 which can be expanded to show detailed results of each simulation run in Figure 7.20. Figure 7.21, 7.22 and Figure 7.23 show the change chart tables for the set-out, transportation, and set-in factors respectively.



**Figure 7.19 Screen shot final results window**

The screenshot shows a window titled ".Models.Frame 1. ReschedulingResults" with a menu bar (File, Edit, Format, Navigate, View, Tools, Help) and a toolbar. Below the toolbar is a table with the following data:

|        | integer<br>1     | time<br>2    | real<br>3          | time<br>4               |
|--------|------------------|--------------|--------------------|-------------------------|
| string | Rescheduling Run | Makespan     | Sequence deviation | Starting time deviation |
| 1      | 4                | 6:40:00.0000 | 40.00              | 14:15.0000              |
| 2      | 1                | 7:10:00.0000 | 10.00              | 15:33.0000              |
| 3      | 2                | 6:55:00.0000 | 20.00              | 15:42.0000              |
| 4      | 3                | 6:55:00.0000 | 30.00              | 16:03.0000              |
| 5      | 5                | 6:40:00.0000 | 40.00              | 14:15.0000              |

**Figure 7.20 Screen shot for detailed rescheduling results**

The screenshot shows a software window with a menu bar (File, Edit, Format, Navigate, View, Tools, Help) and a toolbar. Below the toolbar is a text area containing '10:00.0000'. The main area is a table with the following data:

|        | string<br>0 | time<br>1  | time<br>2  | time<br>3  | time<br>4  |
|--------|-------------|------------|------------|------------|------------|
| string |             | Machine1   | Machine2   | Machine3   | Machine4   |
| 1      | Job1        | 10:00.0000 | 15:00.0000 | 20:00.0000 | 20:00.0000 |
| 2      | Job2        | 10:00.0000 | 10:00.0000 | 20:00.0000 | 20:00.0000 |
| 3      | Job3        | 15:00.0000 | 10:00.0000 | 5:00.0000  | 5:00.0000  |
| 4      | Job4        | 10:00.0000 | 20:00.0000 | 10:00.0000 | 20:00.0000 |
| 5      | Job5        | 10:00.0000 | 20:00.0000 | 20:00.0000 | 20:00.0000 |
| 6      | Job6        | 15:00.0000 | 15:00.0000 | 5:00.0000  | 25:00.0000 |

Figure 7.21 Screen shot for change chart – transportation factor

The screenshot shows a software window with a menu bar (File, Edit, Format, Navigate, View, Tools, Help) and a toolbar. Below the toolbar is a text area containing '10:00.0000'. The main area is a table with the following data:

|        | string<br>0 | time<br>1  | time<br>2  | time<br>3  | time<br>4  |
|--------|-------------|------------|------------|------------|------------|
| string |             | Machine1   | Machine2   | Machine3   | Machine4   |
| 1      | Job1        | 10:00.0000 | 5:00.0000  | 10:00.0000 | 5:00.0000  |
| 2      | Job2        | 5:00.0000  | 5:00.0000  | 10:00.0000 | 5:00.0000  |
| 3      | Job3        | 10:00.0000 | 5:00.0000  | 5:00.0000  | 5:00.0000  |
| 4      | Job4        | 5:00.0000  | 10:00.0000 | 15:00.0000 | 10:00.0000 |
| 5      | Job5        | 10:00.0000 | 10:00.0000 | 10:00.0000 | 10:00.0000 |
| 6      | Job6        | 10:00.0000 | 5:00.0000  | 5:00.0000  | 5:00.0000  |

Figure 7.22 Screen shot for change chart – job set out time factor

The screenshot shows a software window with a menu bar (File, Edit, Format, Navigate, View, Tools, Help) and a toolbar. Below the toolbar is a text area containing '10:00.0000'. The main area is a table with the following data:

|        | string<br>0 | time<br>1  | time<br>2  | time<br>3  | time<br>4  |
|--------|-------------|------------|------------|------------|------------|
| string |             | Machine1   | Machine2   | Machine3   | Machine4   |
| 1      | Job1        | 10:00.0000 | 15:00.0000 | 10:00.0000 | 5:00.0000  |
| 2      | Job2        | 10:00.0000 | 10:00.0000 | 15:00.0000 | 10:00.0000 |
| 3      | Job3        | 10:00.0000 | 10:00.0000 | 5:00.0000  | 5:00.0000  |
| 4      | Job4        | 10:00.0000 | 5:00.0000  | 10:00.0000 | 15:00.0000 |
| 5      | Job5        | 5:00.0000  | 5:00.0000  | 10:00.0000 | 10:00.0000 |
| 6      | Job6        | 15:00.0000 | 15:00.0000 | 5:00.0000  | 10:00.0000 |

Figure 7.23 Screen shot for change chart – job set in time factor

## 7.6.2 Selective re-routing system

The selective rerouting system is implemented similarly as the match-up rescheduling system. As explained, the user has the possibility to select the method of change management. In the following sections, first some customization issues are discussed, and how they were solved. The specifics of the implementation are then given in the following sections.

### 7.6.2.1 Customization of eM-Plant

There were very few modifications done to customize *eM-Plant* for this problem. Most of the components of the methods, tables and variables remain the same for this system as that for the match-up rescheduling system. The same tables for instance serve to carry the data on bounds, starting times, etc for this method too. The only additions were the checkbox for selecting this method of change management shown as *Selective Rerouting* in Figure 7.18. Note that as compared to the Match-up Rescheduling system, the selective rerouting method does not have several options of optimizing Key Performance Indicators. This is because the system inherently optimizes on sequence deviations and not starting time deviations. Also the reactive FAM and ASA are not required to be implemented in this system because they are not required according to the principles of working of this method. **The reader is advised to refer to Appendix 4 for the methods, tables and variables description during the reading of the next section.**

### 7.6.2.2 Workflow of the whole model

The workflow of this system is structured in the same way as it was for the Match-up Rescheduling system and is hence not shown separately. If the user chooses to calculate the rescheduling actions with the selective rerouting algorithm, the *SelectiveReroutingAlgorithm* - method, which implements the functionality described in earlier chapters, is started by the *init* - method. The selective rerouting algorithm uses the *DisturbanceLocation* - variable, the job starting times stored in the *Jobs\_Stage\_Machine* - tables and the processing times in the *Times\_Stage* - tables in order to calculate the time point the first machine becomes idle on the stage where the disturbance occurred. With this time point the algorithm generates the set of candidate jobs for rescheduling and stores them in the *JobsForRerouting* - table. The number of jobs in this set also gives the number of rescheduling alternatives which will be calculated in the following way. This value is stored in the *NumberOfReschedulingRuns* - variable, which serves as a control variable to keep track of the already calculated alternatives. Finally the *SelectiveReroutingAlgorithm* - method reroutes the last job of the candidate set to a new machine and writes the thereby changed routings to the *ReschedulingRoutings* - table. After the completion of the calculations a run of the simulation software is started using the job sequence



from the *DeliveryTable* - table and the routings from the *ReschedulingRoutings* - table. During the run the time points when the different jobs start processing on the machines is recorded by the *StartTimeRecorder* - method in the *ReschedulingJobStartingTimes* - table, while the *JobExitRecorder* - method stores the time points the jobs leave the system in the *ReschedulingJobExitTimes* - table.

After the simulation run is finished the *CalculateReschedulingResults* - method is called, which computes the lead time, sequence stability, and starting time deviation of the current rescheduling alternative using the *Jobs\_Stage\_Machine* - tables, the *FAM\_JobStartingTimes* - table, and the *ReschedulingJobStartingTimes* - table. The results are stored in the *ReschedulingResults* - table. Finally the *endsim* - method is started which checks whether all rescheduling alternatives have already been calculated. If not, the whole process is started anew by calling the *init* - method. Otherwise the *ShowResults* - method is called, which opens the *ResultsDialog* - object, that presents the rescheduling results to the user using the *ReschedulingResults* - table, thus marking the end of the whole rescheduling process. The only difference in the workflow of this system would be the replacement of the type of rescheduling algorithm as seen in Figure 7.13 and no use of the Adaptation Synchrony Analysis System and the Reactive FAM System.

### **7.6.2.3 Running the simulation and the model**

The system also works using the similar Graphical User Interface (GUI) modules as the Match-up rescheduling system. This is obvious since both the Match-up and the Selective rerouting system have the same GUI control and are activated and worked upon using the same set of system components. The system can be run using the main GUI shown in Figure 7.18.

## **7.7 Conclusions**

In this chapter we have discussed the integrated implementation of the optimization functions for scheduling and rescheduling within the simulation system. The predictive scheduling system was implemented to work as a two phased system capable of providing optimization as well as scheduling solutions to the complex production system configuration. The rescheduling system was implemented to provide performance indicators as well as to provide post rescheduling schedule analysis and the Adaptation Synchrony Analysis (ASA) - for the first time, though as a small contribution. In the next chapter, the detailed testing of the systems developed here is described.

# Chapter 8 Quantitative assessment of approaches

## 8.1 Introduction

In this chapter, we discuss the tests conducted for verifying the validity of the developed approaches. Specifically, this chapter is divided into two sections. The first section discusses briefly the layout of the tests and test parameters for the evaluating the predictive system followed by results. The second section presents the same for the reactive system followed by results.

## 8.2 Testing predictive scheduling system: Parameters and tests

The aims of all the tests provided in this section is to test the behaviour of the system under a wide range of operating conditions, and to prove that the systems developed work reasonably well in these conditions. Several tests were carried out to test the effectiveness of the developed predictive scheduling approaches. Table 8.1 shows the test parameters and table 8.2 shows how these parameters were tested. As seen in the test plan, test comparisons are made to check the effect of varying parameters. Each test comparison is designed to give a conclusion for the next tests. Test 1 and 2 test the effect of system size on the computation times required. Here a conclusion about the effect of number of jobs and number of machines on computation times is sought. Test 3 and 4 test the ability of the system to meet delivery times and consider other details of the system altogether under some varying conditions. Test 5 and Test 6 test the effect of increasing job processing times variation with lesser constrained system in terms of buffer and machine configuration. Similarly, Test 7 and Test 8 test for the effect of job processing time variation, with tight buffer capacities and machine configuration. Test 5 and 6 conclude about the effect of processing times, while test 7 and 8 test the same in addition to testing the effect of system configuration in terms of buffer sizing and number of machines. Test 9 uses greater number of jobs with everything else held similar as test 7 – thus providing insight on the effect of number of jobs used in the system with tighter system constraints. Test 10 tests the effects of variation of processing times on a greater number of jobs and compares it with Test 9. All the following tests were made on an Intel Celeron Processor, 1.06 GHz, and 256 MB RAM machine run using WinXP operating system, and *eM-Plant* simulation software.

**Table 8.1 Test parameters**

| <b>Parameters</b> | <b>Description</b>                             |
|-------------------|--|
| Parameter 1       | System size (jobs x machines)                  |
| Parameter 2       | Delivery time for standard and/or special jobs |
| Parameter 3       | Processing times of jobs                       |
| Parameter 4       | Machine configuration                          |
| Parameter 5       | Buffer sizes                                   |
| Parameter 6       | Number of jobs                                 |

**Table 8.2 Test plan and relation to parameters**

| <b>Test comparison</b> | <b>Testing the effect of parameter</b> | <b>Relation ships with other tests</b>  |
|------------------------|--|---|
| Test 1, Test 2         | System size                            | Test 1 and 2 correspond to each other except system size  |
| Test 3, Test 4         | Delivery times                         | Test 3 and 4 correspond to each other except having more special jobs   |
| Test 5, Test 6         | Processing times of jobs               | Test 5 and 6 correspond to each other except processing times   |
| Test 7, Test 8         | Processing times of jobs               | Test 5 and 6 corresponds to 7 and 8 in Number of jobs, Rule generators. Differences in Buffer sizes and Machine configuration |
| Test 7, Test 9         | Number of jobs                         | Test 9 corresponds to test 7 in all parameters except number of jobs  |
| Test 9, Test 10        | Processing times of jobs               | Test 9 corresponds to test 10 in all parameters except processing times of jobs   |

### **8.2.1 Computational times using simulation and optimization based predictive scheduling system**

#### **8.2.1.1 Test 1 data, results and discussions**

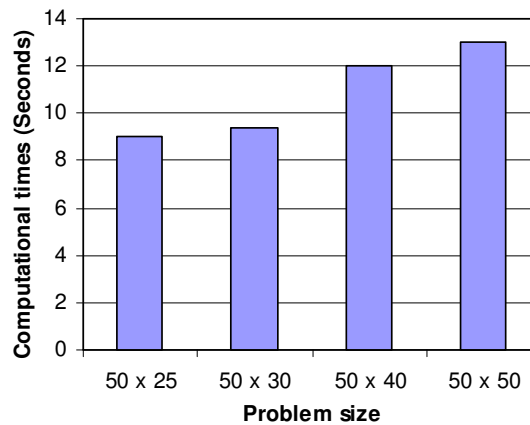
Table 8.3 shows the data used for this test. The aim of this test is to check the total amount of computational time the system needs (optimization plus simulation) with increasing system size which is represented by number of jobs x

number of machines.

**Table 8.3 Data for test case 1**

| System size | Nr. of jobs | Nr. of machines | Nr. of stages | Job processing time variation |
|-------------|-------------|-----------------|---------------|-------------------------------|
| 50 x 25     | 50          | 25              | 5             | 5 - 240                       |
| 50 x 30     | 50          | 30              | 5             | 5 - 240                       |
| 50 x 40     | 50          | 40              | 6             | 5 - 240                       |
| 50 x 50     | 50          | 50              | 6             | 5 - 240                       |

Figure 8.1 shows the resulting computational times by using the system. Specifically worth noting is that computational times last only several seconds. As seen with the same number of jobs, and a higher number of machines, the increase in computation time is not proportionate. When the number of machines are doubled, the computation time increases only marginally. Using a pure simulation based method for deriving an appropriate schedule would certainly take much longer – that too without consider all the other details and constraints of the problem. The computational times shown in Figure 8.1 are obtained using the optimization algorithm plus 2 simulation runs, one for simulating the result of the optimization algorithm, and the second for analysing this result using the FAM.



**Figure 8.1 Computational times for the predictive scheduling system**

### 8.2.1.2 Test data 2, results and discussions

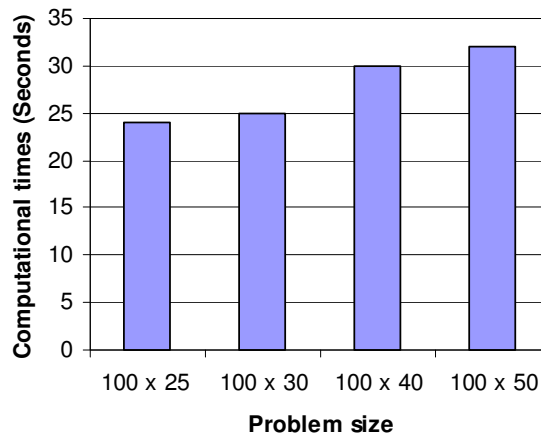
Table 8.4 shows the data used for the second test with even bigger problem sizes. The data is arranged in a similar way as for test 1. This time 100 jobs were used with increasing machine sizes. Figure 8.2 shows the results on computational times obtained for this test. It can be seen that here too we have a similar pattern of

the computation times required. Increasing number of jobs to 100 and increasing the machine sizes and stages for each setting, cause the times to increase only marginally.

**Table 8.4 Data for test case 2**

| <b>Problem size</b> | <b>Nr. of jobs</b> | <b>Nr. of machines</b> | <b>Nr. of stages</b> | <b>Job processing time variation</b> |
|---------------------|--------------------|------------------------|----------------------|--------------------------------------|
| 100 x 25            | 100                | 25                     | 5                    | 5 - 240                              |
| 100 x 30            | 100                | 30                     | 5                    | 5 - 240                              |
| 100 x 40            | 100                | 40                     | 6                    | 5 - 240                              |
| 100 x 50            | 100                | 50                     | 6                    | 5 - 240                              |

When test 2 is compared to test 1, it can be seen that within these two tests, the trend is the same, but that the number of jobs (and not the number of machines) play a major role in determining computation times. In test 1, using a system size of 50 x 25 resulted in a computation time of 9 seconds, while in test 2, a system size of 100 x 25 results in more than 2 fold increase in computation times. A further test was carried out to confirm the claims, with 20 stages, with 6 machines (total 120 machines) each and 100 jobs. For this test configuration (100 x 120), the computation times were obtained as 5.5 minutes. So it seems that along with the number of jobs, if the number of machines are doubled, it also leads to a big increase in computation times. From the the results of all these tests, one may infer that a system size of 500 x 100 would result in computation times of approximately less than 30 minutes. Note that the times required for the simulation runs are with the animation of the simulation turned on. When animations are turned off, even shorter times for simulation can be obtained.



**Figure 8.2 Computational times for the predictive scheduling system**

## 8.2.2 Delivery time optimization results

### 8.2.2.1 Test 3 data, results and discussions

Table 8.5 shows the data used for this test. Other data was also used on requirements for delivery and job routings within the system as seen in Figure 8.3. Three scenarios were tested namely jobs with special routes only, jobs with no special routes but with delivery times, and jobs with both delivery times and routing constraints. Similarly, unavailability of machines, materials and tools as shown in Figures 8.4 and 8.5 were also set. Each of them show the times when the machines or resources will be available. This test aims to test the effect of including all these constraints together alongwith buffers and testing the entire system for the effectiveness of the simulation based FAM and the optimization system. The tolerance value was kept to 20 minutes. Figure 8.6 shows the results of this test. Jobs 6 and 4, were completed as per the demanded completion time, in *eM-Plant* time format. Figure 8.7 shows the delivery time calculated by the optimization

**Table 8.5 Data used for test case 3**

| Nr. of jobs | Nr. of stages | Nr. of machines                                      | Decision points  | Processing times              | Buffer capacity                      |
|-------------|---------------|--|--|-------------------------------|--------------------------------------|
| 50          | 4             | Stage 1: 5<br>Stage 2: 4<br>Stage 3: 5<br>Stage 4: 4 | Stage 1 to 4:<br>Optimality rule generator for buffers | Varying from 20 to 40 minutes | Varying from 5 to 8 on stages 1 to 4 |

| Job1   | string 1 | string 2        | time 3        | sl 4 |
|--------|----------|-----------------|---------------|------|
| string | JobName  | Path (4 Stages) | Delivery time |      |
| 1      | Job1     |                 |               |      |
| 2      | Job2     | 1,3,4,2         |               |      |
| 3      | Job3     |                 |               |      |
| 4      | Job4     |                 | 5:00:00.0000  |      |
| 5      | Job5     |                 |               |      |
| 6      | Job6     | 2,2,2,2         | 3:00:00.0000  |      |
| 7      | Job7     |                 |               |      |
| 8      | Job8     |                 |               |      |
| 9      | Job9     |                 |               |      |

**Figure 8.3: Setting delivery times for standard and special job flows**

|        | string<br>0     | time<br>1         | time<br>2       | strir<br>3 |
|--------|-----------------|-------------------|-----------------|------------|
| string | Machine name    | Maintenance start | Maintenance end |            |
| 7      | Machine2_Stage2 |                   |                 |            |
| 8      | Machine3_Stage2 |                   |                 |            |
| 9      | Machine4_Stage2 | 1:10:00.0000      | 1:40:00.0000    |            |
| 10     | Machine1_Stage3 |                   |                 |            |
| 11     | Machine2_Stage3 |                   |                 |            |
| 12     | Machine3_Stage3 | 1:40:00.0000      | 2:10:00.0000    |            |
| 13     | Machine4_Stage3 |                   |                 |            |
| 14     | Machine5_Stage3 |                   |                 |            |
| 15     | Machine1_Stage4 |                   |                 |            |
| 16     | Machine2_Stage4 |                   |                 |            |

Figure 8.4: Setting machine maintenance times

|        | string<br>1 | time<br>2    | strir<br>3 |
|--------|-------------|--------------|------------|
| string | Job1        | Available at |            |
| 1      | Job1        |              |            |
| 2      | Job2        |              |            |
| 3      | Job3        | 40:00.0000   |            |
| 4      | Job4        |              |            |
| 5      | Job5        |              |            |
| 6      | Job6        |              |            |
| 7      | Job7        |              |            |
| 8      | Job8        | 1:00:00.0000 |            |
| 9      | Job9        |              |            |
| 10     | Job10       |              |            |
| 11     | Job11       |              |            |
| 12     | Job12       |              |            |
| 13     | Job13       | 20:00.0000   |            |
| 14     | Job14       |              |            |

Figure 8.5: Setting material availability

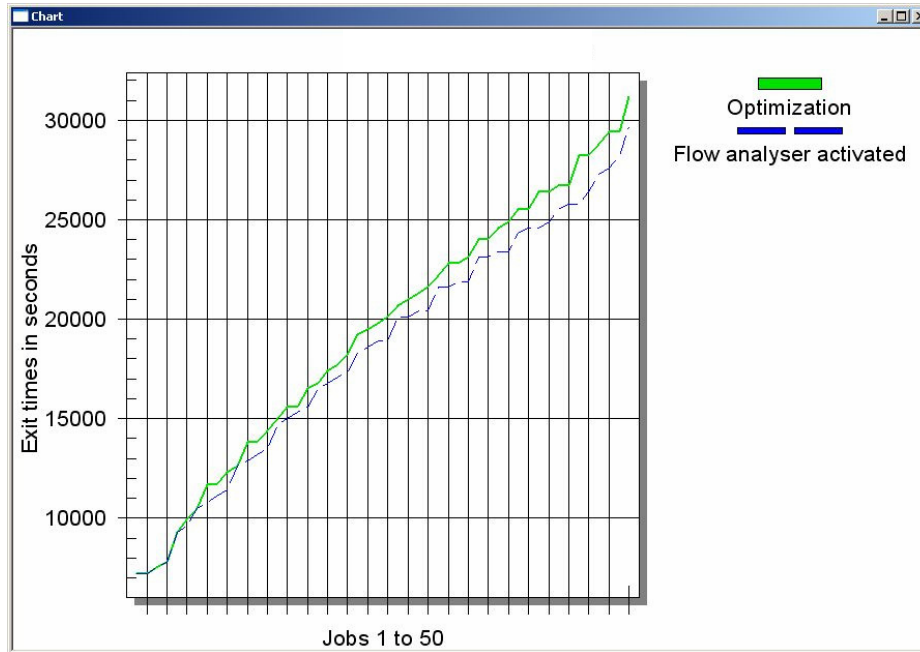
|        | time<br>2    | time<br>3               | string<br>6            | string<br>7                     |
|--------|--------------|-------------------------|------------------------|---------------------------------|
| string | Optimization | Flow analyser activated | Jobs from optimization | Jobs from flow analyser activat |
| 4      | 2:10:00.0000 | 2:10:00.0000            | Job15                  | Job15                           |
| 5      | 2:35:00.0000 | 2:35:00.0000            | Job6                   | Job6                            |
| 6      | 2:45:00.0000 | 2:40:00.0000            | Job49                  | Job38                           |
| 7      | 2:55:00.0000 | 2:55:00.0000            | Job22                  | Job10                           |
| 8      | 3:15:00.0000 | 3:00:00.0000            | Job38                  | Job22                           |
| 9      | 3:15:00.0000 | 3:05:00.0000            | Job25                  | Job49                           |
| 10     | 3:25:00.0000 | 3:10:00.0000            | Job10                  | Job25                           |
| 11     | 3:30:00.0000 | 3:30:00.0000            | Job27                  | Job9                            |
| 12     | 3:50:00.0000 | 3:35:00.0000            | Job39                  | Job27                           |
| 13     | 3:50:00.0000 | 3:40:00.0000            | Job34                  | Job43                           |
| 14     | 4:00:00.0000 | 3:45:00.0000            | Job21                  | Job36                           |
| 15     | 4:10:00.0000 | 4:05:00.0000            | Job36                  | Job21                           |
| 16     | 4:20:00.0000 | 4:10:00.0000            | Job42                  | Job39                           |
| 17     | 4:20:00.0000 | 4:15:00.0000            | Job43                  | Job41                           |
| 18     | 4:35:00.0000 | 4:20:00.0000            | Job41                  | Job34                           |
| 19     | 4:40:00.0000 | 4:35:00.0000            | Job4                   | Job23                           |
| 20     | 4:50:00.0000 | 4:40:00.0000            | Job1                   | Job4                            |
| 21     | 4:55:00.0000 | 4:45:00.0000            | Job9                   | Job4                            |

Figure 8.6: Resulting plan and schedules obtained with both systems

|        | time<br>1     | object<br>2 | integer<br>3 | string<br>4 | table<br>5 |
|--------|---------------|-------------|--------------|-------------|------------|
| string | Delivery Time | MU          | Number       | Name        | Attribute  |
| 39     | 0.0000        | MUs.Job47   | 1            | Job47       |            |
| 40     | 0.0000        | MUs.Job48   | 1            | Job48       |            |
| 41     | 0.0000        | MUs.Job7    | 1            | Job7        |            |
| 42     | 0.0000        | MUs.Job17   | 1            | Job17       |            |
| 43     | 0.0000        | MUs.Job46   | 1            | Job46       |            |
| 44     | 0.0000        | MUs.Job19   | 1            | Job19       |            |
| 45     | 0.0000        | MUs.Job32   | 1            | Job32       |            |
| 46     | 0.0000        | MUs.Job37   | 1            | Job37       |            |
| 47     | 0.0000        | MUs.Job2    | 1            | Job2        |            |
| 48     | 20:00.0000    | MUs.Job13   | 1            | Job13       |            |
| 49     | 50:00.0000    | MUs.Job3    | 1            | Job3        |            |
| 50     | 1:00:00.0000  | MUs.Job8    | 1            | Job8        |            |
| 51     |               |             |              |             |            |

Figure 8.7: Job delivery table calculated by algorithm





**Figure 8.8: Make span comparing pure optimization and optimization with FAM schedules**

algorithm, which shows the sequence in which jobs enter the system. As seen, jobs 3 and 8 were started at the appropriate times due to delayed arrival of materials and supplies. When the FAM was used with the optimality rule for buffers, the schedule was better by about 5 % in make-span. Table 8.6 shows the comparison of the demanded delivery time (negative values indicate there was no lateness), with the delivery times obtained using the optimization and the optimization with simulation based FAM.

**Table 8.6 Lateness measurements  $L_j$  for Test 3**

| Jobs  | $C_j$        |                      | $d_j$         | $L_j = C_j - d_j$ |                      |
|-------|--------------|----------------------|---------------|-------------------|----------------------|
|       | Optimization | Simulation based FAM | Delivery time | Optimization      | Simulation based FAM |
| Job 4 | 4:55.00.00   | 4:45.00.00           | 5:00.00.00    | -5.00.00          | -15.00.00            |
| Job 6 | 2:35.00.00   | 2:35.00.00           | 3:00.00.00    | -25.00.00         | -25.00.00            |

### 8.2.2.2 Test 4 data, results and discussions

Table 8.7 shows the data used for the test case 4 which aims at measuring the efficiency of the predictive scheduling system in terms of meeting delivery

deadlines for a greater number of jobs with special requirements and delivery time requirements. Figure 8.9 shows the routing and delivery time constraints. Job 5 is seen to have a special path shown by 3,4,4,1 (meaning job 1 travels to machine 3 on stage 1, to machine 4 on stage 2, to machine 4 on stage 3 and to machine 1 on stage 4) due to constraints on manufacturing technologies that can be used to manufacture it. Similarly job 9 has routing constraints shown as path 1,2,2,4. All other jobs only have delivery time requirements with no restrictions on routing. Note that job 1 and job 10 have the same delivery time requirements.

**Table 8.7 Data used for test case 4**

| Nr. of jobs | Nr. Of stages | Nr. of machines                                      | Decision points   | Processing times                    | Buffer capacity    |
|-------------|---------------|--|---|-------------------------------------|--------------------|
| 50          | 4             | Stage 1: 5<br>Stage 2: 5<br>Stage 3: 5<br>Stage 4: 5 | Stage 1 to 4:<br>Optimality rule<br>generator for buffers | Varying<br>from 10 to<br>25 minutes | 8 at all<br>stages |

| string | string 1 | string 2        | time 3        |
|--------|----------|-----------------|---------------|
| string | JobName  | Path (4 Stages) | Delivery time |
| 1      | Job1     |                 | 2:00:00.0000  |
| 2      | Job2     |                 | 2:15:00.0000  |
| 3      | Job3     |                 | 2:30:00.0000  |
| 4      | Job4     |                 | 2:45:00.0000  |
| 5      | Job5     | 3,4,4,1         | 3:00:00.0000  |
| 6      | Job6     |                 | 3:15:00.0000  |
| 7      | Job7     |                 | 3:30:00.0000  |
| 8      | Job8     |                 | 3:45:00.0000  |
| 9      | Job9     | 1,2,2,4         | 4:00:00.0000  |
| 10     | Job10    |                 | 2:00:00.0000  |

**Figure 8.9 Delivery constraints and job paths set by user for special jobs**

Figure 8.10a and 8.10b, show the delivery times and final results (for all 50 jobs in 2 parts) after running the algorithm and system. These special jobs are marked as ellipses on these figures. As seen in the figures, all but Job 9 are

delivered earlier than their times. Job 9 is analyzed in Figure 8.10a, and it is seen that using the optimization algorithm, it is delivered on time at 3:20.00.00. However, due to the flow analyser system, it is delayed by 5 minutes seen as dotted ellipse in Figure 8.10b. This delay of 5 minutes is due to the overtakings of jobs in sequence during the flow analysis step using simulation. Refer to Table 8.8 for the lateness measurements for this test.

| string | time 2       | time 3                  | string 6            | string 7                        |
|--------|--------------|-------------------------|---------------------|---------------------------------|
|        | Heuristic    | Flow analyser activated | Jobs from heuristic | Jobs from flow analyser activat |
| 1      | 45:00.0000   | 45:00.0000              | Job40               | Job40                           |
| 2      | 50:00.0000   | 50:00.0000              | Job33               | Job33                           |
| 3      | 1:10:00.0000 | 1:05:00.0000            | Job15               | Job1                            |
| 4      | 1:10:00.0000 | 1:10:00.0000            | Job21               | Job15                           |
| 5      | 1:10:00.0000 | 1:10:00.0000            | Job42               | Job21                           |
| 6      | 1:15:00.0000 | 1:10:00.0000            | Job14               | Job42                           |
| 7      | 1:20:00.0000 | 1:15:00.0000            | Job1                | Job14                           |
| 8      | 1:30:00.0000 | 1:30:00.0000            | Job31               | Job31                           |
| 9      | 1:35:00.0000 | 1:35:00.0000            | Job10               | Job10                           |
| 10     | 1:35:00.0000 | 1:35:00.0000            | Job38               | Job38                           |
| 11     | 1:35:00.0000 | 1:35:00.0000            | Job2                | Job2                            |
| 12     | 1:50:00.0000 | 1:45:00.0000            | Job19               | Job18                           |
| 13     | 1:55:00.0000 | 1:45:00.0000            | Job32               | Job3                            |
| 14     | 2:00:00.0000 | 1:50:00.0000            | Job18               | Job19                           |
| 15     | 2:00:00.0000 | 2:00:00.0000            | Job3                | Job32                           |
| 16     | 2:05:00.0000 | 2:05:00.0000            | Job13               | Job26                           |
| 17     | 2:05:00.0000 | 2:05:00.0000            | Job22               | Job22                           |
| 18     | 2:10:00.0000 | 2:05:00.0000            | Job4                | Job4                            |
| 19     | 2:25:00.0000 | 2:15:00.0000            | Job24               | Job24                           |
| 20     | 2:30:00.0000 | 2:20:00.0000            | Job26               | Job13                           |
| 21     | 2:30:00.0000 | 2:25:00.0000            | Job5                | Job6                            |
| 22     | 2:35:00.0000 | 2:35:00.0000            | Job6                | Job16                           |
| 23     | 2:40:00.0000 | 2:35:00.0000            | Job16               | Job5                            |
| 24     | 2:45:00.0000 | 2:35:00.0000            | Job11               | Job11                           |
| 25     | 2:50:00.0000 | 2:40:00.0000            | Job7                | Job43                           |

**Figure 8.10a Resulting schedule obtained with optimization and simulation based FAM: Part 1**

One can conclude from this that the heuristic (the optimization algorithm results

after 1 normal simulation run) delivers and schedules jobs according to their delivery times (with minimum or no lateness), but when the simulation based FAM is used, it is possible that one or a few jobs are delivered slightly late. Note in Table 8.8, that jobs were finished at the same time or earlier than calculated by the algorithm, by the FAM in 8 cases, while in 2 cases, they were delivered at the same time or late. As earlier, the minus number for lateness in Table 8.8 indicate that the jobs finished earlier by that amount. When test 3 and 4 were compared to each other, it can be seen that when more jobs are considered to have special requirements, the developed application still works well.

| string | time 2       | time 3                  | string 6            | string 7                        |
|--------|--------------|-------------------------|---------------------|---------------------------------|
|        | Heuristic    | Flow analyser activated | Jobs from heuristic | Jobs from flow analyser activat |
| 26     | 2:55:00.0000 | 2:45:00.0000            | Job43               | Job12                           |
| 27     | 2:55:00.0000 | 2:45:00.0000            | Job36               | Job7                            |
| 28     | 2:55:00.0000 | 2:50:00.0000            | Job39               | Job8                            |
| 29     | 3:05:00.0000 | 2:55:00.0000            | Job23               | Job39                           |
| 30     | 3:05:00.0000 | 2:55:00.0000            | Job8                | Job23                           |
| 31     | 3:10:00.0000 | 3:05:00.0000            | Job45               | Job45                           |
| 32     | 3:15:00.0000 | 3:05:00.0000            | Job44               | Job36                           |
| 33     | 3:15:00.0000 | 3:10:00.0000            | Job12               | Job48                           |
| 34     | 3:20:00.0000 | 3:10:00.0000            | Job9                | Job27                           |
| 35     | 3:25:00.0000 | 3:15:00.0000            | Job48               | Job44                           |
| 36     | 3:25:00.0000 | 3:20:00.0000            | Job25               | Job28                           |
| 37     | 3:30:00.0000 | 3:20:00.0000            | Job27               | Job20                           |
| 38     | 3:30:00.0000 | 3:25:00.0000            | Job20               | Job30                           |
| 39     | 3:35:00.0000 | 3:25:00.0000            | Job28               | Job25                           |
| 40     | 3:45:00.0000 | 3:30:00.0000            | Job34               | Job34                           |
| 41     | 3:45:00.0000 | 3:35:00.0000            | Job30               | Job37                           |
| 42     | 3:45:00.0000 | 3:35:00.0000            | Job49               | Job49                           |
| 43     | 3:45:00.0000 | 3:40:00.0000            | Job37               | Job41                           |
| 44     | 3:50:00.0000 | 3:40:00.0000            | Job35               | Job29                           |
| 45     | 3:55:00.0000 | 3:50:00.0000            | Job50               | Job46                           |
| 46     | 4:00:00.0000 | 3:50:00.0000            | Job41               | Job35                           |
| 47     | 4:00:00.0000 | 3:50:00.0000            | Job46               | Job50                           |
| 48     | 4:00:00.0000 | 3:55:00.0000            | Job17               | Job17                           |
| 49     | 4:00:00.0000 | 4:00:00.0000            | Job29               | Job47                           |
| 50     | 4:10:00.0000 | 4:05:00.0000            | Job47               | Job9                            |

**Figure 8.10b Resulting schedule obtained with optimization and simulation based FAM: Part 2**

**Table 8.8 Lateness measurements  $L_j$  for Test 4**

| Jobs   | $C_j$        |                      | $d_j$         | $L_j = C_j - d_j$ |                      |
|--------|--------------|----------------------|---------------|-------------------|----------------------|
|        | Optimization | Simulation based FAM | Delivery time | Optimization      | Simulation based FAM |
| Job 1  | 1:20.00.00   | 1:05.00.00           | 2:00.00.00    | -40.00.00         | -55.00.00            |
| Job 2  | 1:35.00.00   | 1:35.00.00           | 2:15.00.00    | -40.00.00         | -40.00.00            |
| Job 3  | 2:00.00.00   | 1:45.00.00           | 2:30.00.00    | -30.00.00         | -45.00.00            |
| Job 4  | 2:10.00.00   | 2:05.00.00           | 2:45.00.00    | -35.00.00         | -40.00.00            |
| Job 5  | 2:30.00.00   | 2:35.00.00           | 3:00.00.00    | -30.00.00         | -25.00.00            |
| Job 6  | 2:35.00.00   | 2:25.00.00           | 3:15.00.00    | -40.00.00         | -50.00.00            |
| Job 7  | 2:50.00.00   | 2:45.00.00           | 3:30.00.00    | -40.00.00         | -45.00.00            |
| Job 8  | 3:03.00.00   | 2:50.00.00           | 3:45.00.00    | -42.00.00         | -55.00.00            |
| Job 9  | 3:20.00.00   | 4:05.00.00           | 4:00.00.00    | -40.00.00         | 5.00.00              |
| Job 10 | 1:35.00.00   | 1:35.00.00           | 2:00.00.00    | -25.00.00         | -25.00.00            |

### 8.2.3 Test 5 data, results and discussions

Table 8.9 shows the data used for test 5. The aim of test 5 is to prove that the simulation based FAM system produces better results than the pure optimization algorithm, when the system is not subject to tighter buffer sizing constraints, using a combination of optimality and validity rule generators. The aim of test 5 (and test 6) is to check the effect of varying processing times while keeping the same number of machines at each stage and a higher level of buffer sizes.

**Table 8.9 Data for test case 5**

| Nr. of jobs | Nr. Of stages | Nr. of machines | Decision points   | Processing times             | Buffer capacity   |
|-------------|---------------|-----------------|---|------------------------------|-------------------|
| 50          | 4             | 4               | Stage 1: Validity rule generator for buffers<br>Stage 2 to 4: Optimality rule generator for buffers | Varying from 5 to 25 minutes | 15 at all buffers |

This means that there are no bottlenecks created due to uneven machine loading between stages and due to the lack of buffer capacities.

#### 8.2.3.1 Makespan comparison for various methods

Figure 8.11 shows the results of the makespan and the times when each job leaves the system after completion. It is observed that random scheduling is much worse in the long run as compared to the optimization algorithm. It can also be

seen that in this test, no reduction in overall makespan was achieved using the simulation based FAM – although the results were not worse than the optimization algorithm.

### 8.2.3.2 Performance benefits of the simulation based FAM system

Regardless of the makespan result, Figure 8.12 shows the percentage reduction in job finishing times (JFT) each job obtained as a result of using the simulation based FAM system.

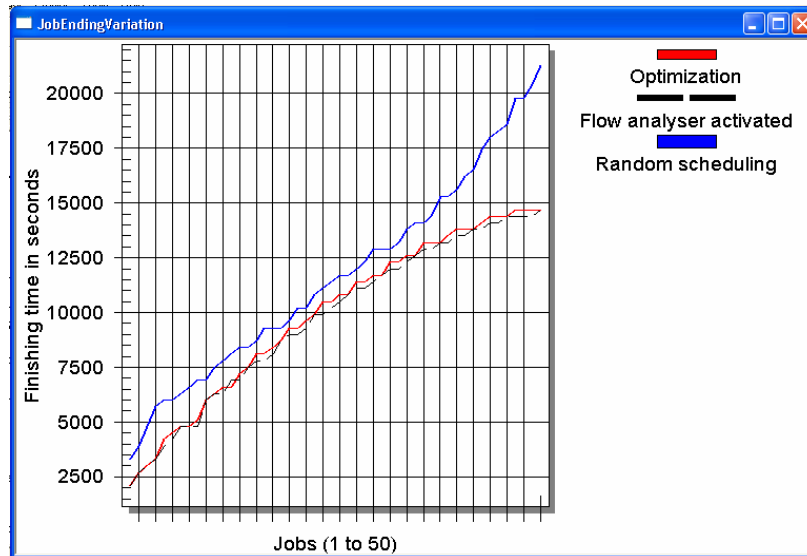


Figure 8.11 Makespan results for different methods of scheduling

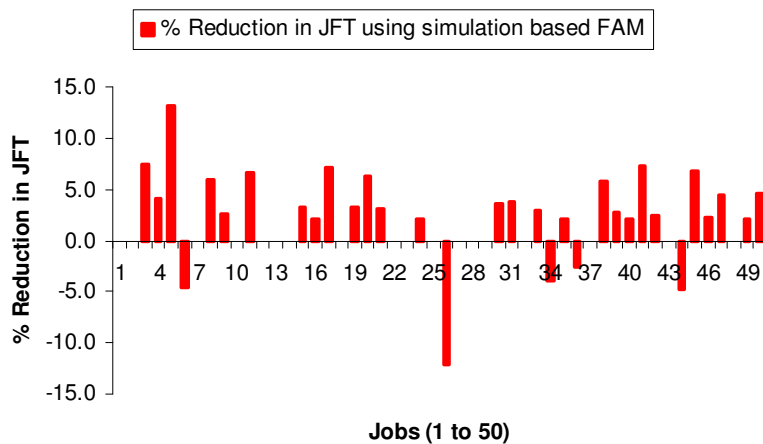


Figure 8.12 Percentage reduction in JFT using simulation based FAM

About 54 % of the jobs finished earlier, while only 10 % finished later than the plan calculated by the optimization algorithm. Other jobs had no effect on their finishing times. Figure A9 in the appendix 5 shows the job finishing times of each of the jobs. To conclude, it seems that when there are no bottlenecks in the system due to different number of machines between stages, and due to the buffer sizing, the simulation based FAM does not improve the makespan – partly due to the effect of processing times and the validity rule generator used at one stage. However, it only improved the job finishing times for about half of the jobs in the system.

#### 8.2.4 Test 6 data, results and discussions

Table 8.10 shows the data used for test 6. The aim of this test was to check the influence of larger variation in job processing times using the same other data used for test 5. As for test 5, here also the effect of not constraining the system with limited buffers and machine configuration is checked.

##### 8.2.4.1 Makespan comparison for various methods

Figure 8.13 shows the result of test 6, which is compared to test 5. As seen result, a much better makespan was achieved. The optimization algorithm provided a makespan of 900 minutes while the simulation based FAM further reduced the makespan to 816 minutes which is equivalent to 10 % lesser than the optimization algorithm. Comparison of the random scheduling process and the optimization suggests an improvement of more than 20 % using the methods developed.

##### 8.2.4.2 Performance benefits of the simulation based FAM system

Figure 8.14 shows the percentage reduction in job finishing times using the simulation based FAM procedure. It can be seen as compared to Figure 8.12 of test 5, that the results are better. Although the number of jobs finishing earlier is less, overall a higher reduction in JFT was obtained. The reason why some jobs have a negative reduction (or an increase) in JFT is because of the fact that some jobs overtake other jobs in the process of the flow analysis process. Figure A10 in the appendix 6 shows the job finish times for the test for all the three methods of scheduling.

**Table 8.10 Data for test case 6**

| Nr. of jobs | Nr. Of stages | Nr. of machines | Decision points                                   | Processing times              | Buffer capacity   |
|-------------|---------------|-----------------|---|-------------------------------|-------------------|
| 50          | 4             | 4               | Stage 1: Validity rule generator for buffers      | Varying from 5 to 100 minutes | 15 at all buffers |
|             |               |                 | Stage 2 to 4: Validity rule generator for buffers |                               |                   |

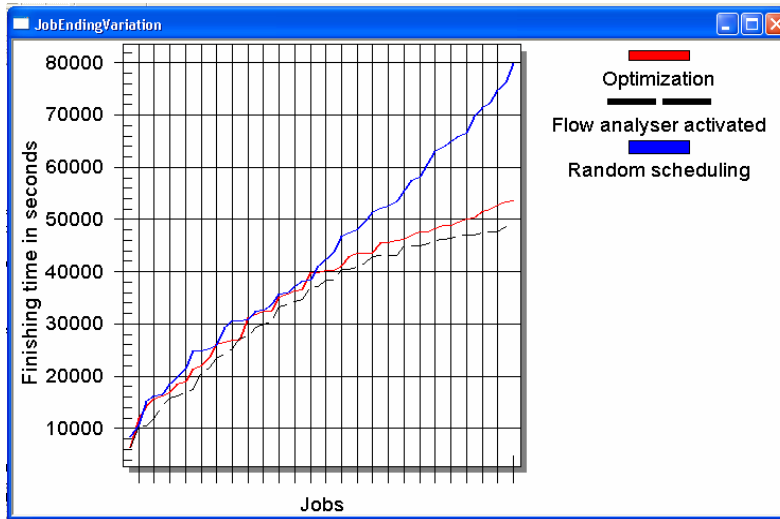


Figure 8.13 Makespan results for different methods of scheduling

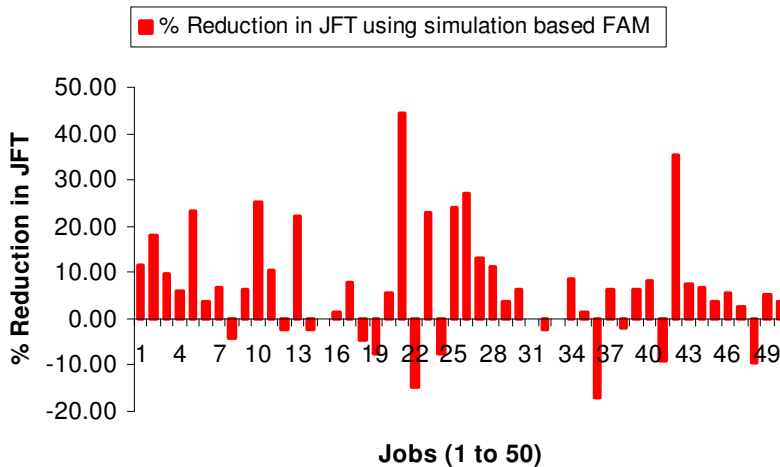


Figure 8.14 Percentage reduction in JFT using simulation based FAM

From the above results, we can conclude that when the system is *not* subject to constraints of buffer sizing and when the processing time variation is higher, the system provides better results in terms of reduction of JFT and makespan.

### 8.2.5 Test 7 data, results and discussions

Table 8.11 shows the data used for test 7. The aim of this test (and test 8) was to determine the influence of job processing times whilst keeping tighter constraints on the number of machines for all stages and the buffer sizing. When compared to test 5 and test 6, here we have lesser machines between stages and



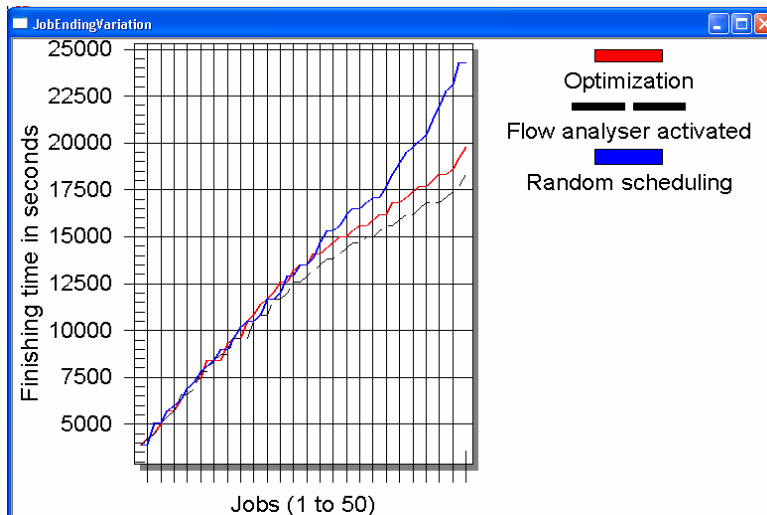
we also have smaller buffer capacities.

**Table 8.11 Data for test case 7**

| Nr. of jobs | Nr. of stages | Nr. of machines                                      | Decision points   | Processing times             | Buffer capacity     |
|-------------|---------------|--|---|------------------------------|---------------------|
| 50          | 4             | Stage 1: 5<br>Stage 2: 3<br>Stage 3: 5<br>Stage 4: 3 | Stage 1: Validity rule generator for buffers<br>Stage 2 to 4: Optimality rule generator for buffers | Varying from 5 to 25 minutes | Varying from 1 to 3 |

### 8.2.5.1 Makespan comparison for different scheduling methods

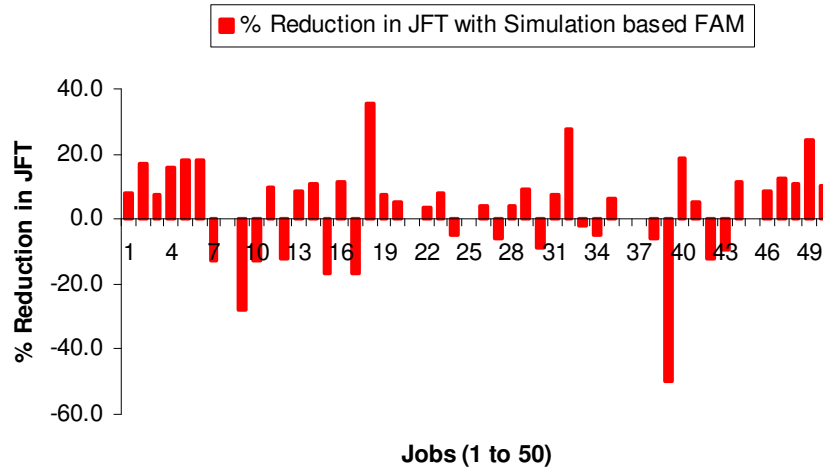
Figure 8.15 shows the makespan comparison for the three methods. It is seen that again a 10 % reduction in makespan was achieved using the simulation based FAM, while the optimization proved again to provide as much as 20 percent reduction in flow times. When comparing this result to Figure 8.11, it can be seen that the simulation based FAM system provides reduced makespan figures when the system is subject to tighter constraints like buffer sizing and machine sizing for similar processing times.



**Figure 8.15: Makespan results for different methods of scheduling**

### 8.2.5.2 Performance benefits of the simulation based FAM system

Figure 8.16 shows the percentage reduction in job finishing times using the simulation based FAM system. 15 jobs had increase in JFT while 19 jobs finished earlier (as compared to 27 jobs in test 5), while the others had no change in reducing the JFT.



**Figure 8.16: Percentage reduction in JFT using simulation based FAM**

### 8.2.6 Test 8 data, results and discussions

Table 8.12 shows the test data used for test 8. The aim of this test is to check the influence of a large variation of processing times, keeping all the other constraints the same as used for test 7.

**Table 8.12 Data for test case 8**

| Nr. of jobs | Nr. Of stages | Nr. of machines                                      | Decision points   | Processing times              | Buffer capacity    |
|-------------|---------------|--|---|-------------------------------|--------------------|
| 50          | 4             | Stage 1: 5<br>Stage 2: 3<br>Stage 3: 5<br>Stage 4: 3 | Stage 1: Validity rule generator for buffers<br>Stage 2 to 4: Optimality rule generator for buffers | Varying from 5 to 100 minutes | Varying fro 1 to 3 |

#### 8.2.6.1 Makespan comparison using different methods

Figure 8.17 shows the comparison of makespan for different methods of scheduling. It can be seen that the random scheduling method has more than 20 % increase in makespan values. There seems no improvement in the makespan using the simulation based FAM system. When this is compared to test 7, one can conclude that the system performs better (in terms of makespan) when the processing time variations are smaller, and when the system is subject to higher constraints in terms of the number of machines available and the buffer sizing.

#### 8.2.6.2 Performance benefits of the simulation based FAM system

Figure 8.18 shows the percentage reduction in JFT using the simulation based FAM system. It can be seen as compared to Figure 8.16 that we have

worse results here too. 18 jobs resulted in higher job finishing times while fewer jobs resulted in reduction of finishing times. When this test is considered as a stand alone test, the simulation based FAM system does not provide worse results in makespan, but improves slightly the job finishing times for about 50 % of the jobs.

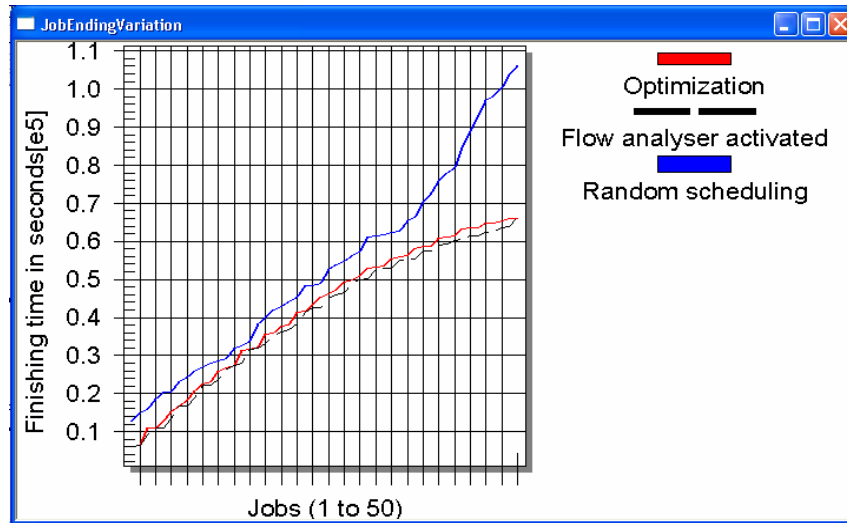


Figure 8.17: Makespan results for different methods of scheduling

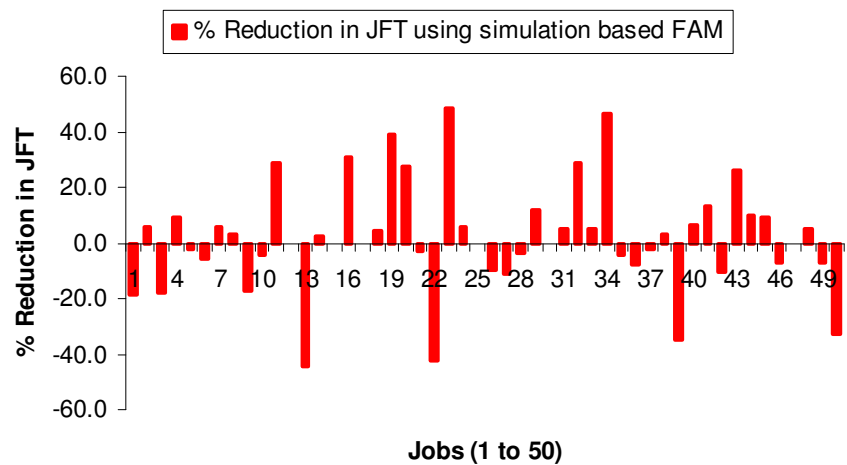


Figure 8.18: Percentage reduction of JFT using simulation based FAM

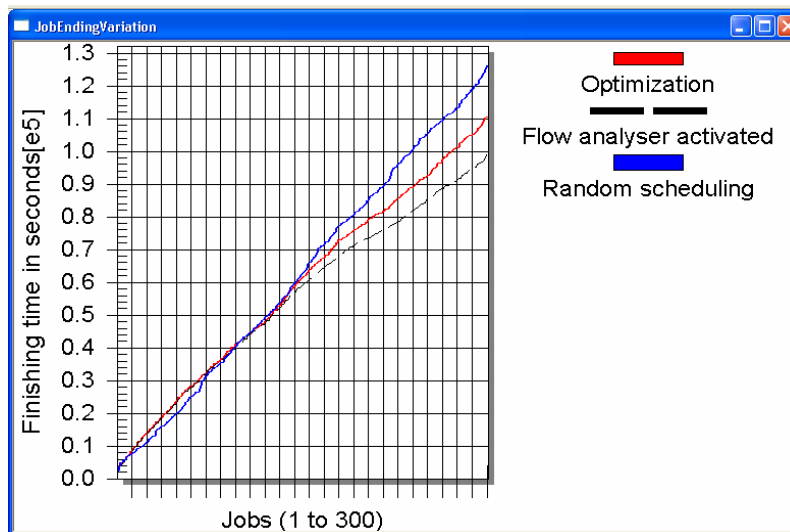
### 8.2.7 Test 9 data, results and discussions

Table 8.13 shows the data used for test case 9. The aim of this test is to prove the applicability of the developed system for a bigger number of jobs and where each job has some variations in processing times, with additional constraints

on the system size.

**Table 8.13 Data for test case 9**

| Nr. of jobs | Nr. Of stages | Nr. of machines                                      | Decision points   | Processing times             | Buffer capacity    |
|-------------|---------------|--|---|------------------------------|--------------------|
| 300         | 4             | Stage 1: 5<br>Stage 2: 3<br>Stage 3: 5<br>Stage 4: 3 | Stage 1: Validity rule generator for buffers<br>Stage 2 to 4: Optimality rule generator for buffers | Varying from 5 to 25 minutes | Varying fro 1 to 3 |



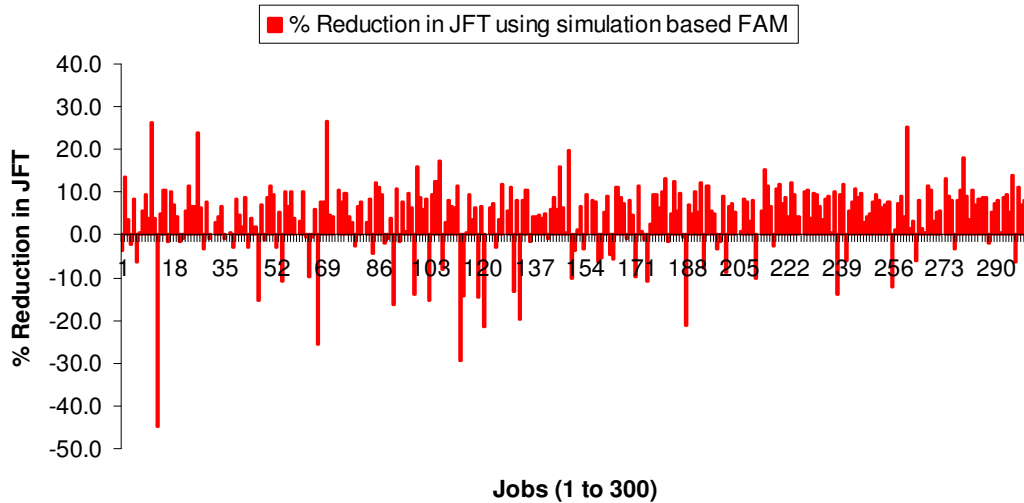
**Figure 8.19 Makespan comparison using different approaches**

### 8.2.7.1 Makespan comparison using different methods

Figure 8.19 shows the makespan obtained using different methods of scheduling. As seen in the long run, the simulation based FAM system provides reduction in makespan figures of about 10 %. This result proves that even for a great number of jobs, the simulation based system provides better overall results.

### 8.2.7.2 Performance benefits of the simulation based FAM system

Figure 8.20 shows the reduction in job finishing times (JFT) using the simulation based FAM system. More than 80 % of the jobs had reduced job finishing times as compared to the predictive schedule of the optimization algorithm. This result proves that the system also reduces the job finishing times for a greater number of jobs in the system.



**Figure 8.20 Percentage reduction in JFT using simulation based FAM – Test 9**

### 8.2.8 Test 10 data, results and discussions

Table 8.14 shows the data used for test 10. The aim of test 10 is to prove the applicability of the system for a larger number of jobs with great variation in processing times. Rest of the parameters were the same as Test 9. Test 10 has the same aim as Test 9, the only difference being different job processing times.

**Table 8.14 Data for test case 10**

| Nr. of jobs | Nr. Of stages | Nr. Of machines                                      | Decision points   | Processing times              | Buffer capacity    |
|-------------|---------------|--|---|-------------------------------|--------------------|
| 300         | 4             | Stage 1: 5<br>Stage 2: 3<br>Stage 3: 5<br>Stage 4: 3 | Stage 1: Validity rule generator for buffers<br>Stage 2 to 4: Optimality rule generator for buffers | Varying from 5 to 100 minutes | Varying fro 1 to 3 |

#### 8.2.8.1 Makespan comparison using different methods

Figure 8.21 shows the makespan obtained for different methods of scheduling. As compared to Figure 8.19, it is seen that lower reduction in makespan value was obtained using the simulation based FAM. This means that when there is a larger variation in processing times in a system with constraints and a larger number of jobs, the system provides improvements, but lesser than if the variation in job processing times are lower.

#### 8.2.8.2 Performance benefits of the simulation based FAM system

Figure 8.22 shows the results of the reductions in JFT obtained using the

simulation based FAM. It is seen that here too about 80 % of the jobs finished earlier. This result is similar to Figure 8.20. To conclude, it seems that the system not only improves system performance for the case with smaller number of jobs, but it does so with bigger number of jobs too.

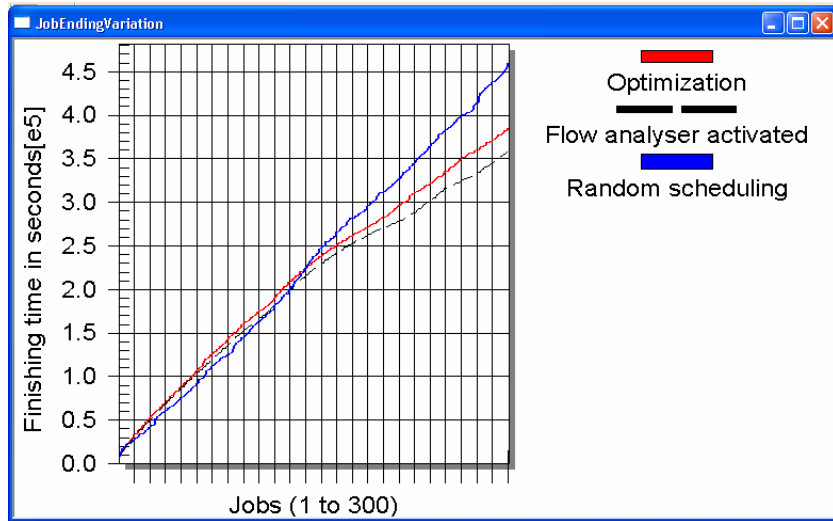


Figure 8.21: Makespan comparison for various methods of scheduling

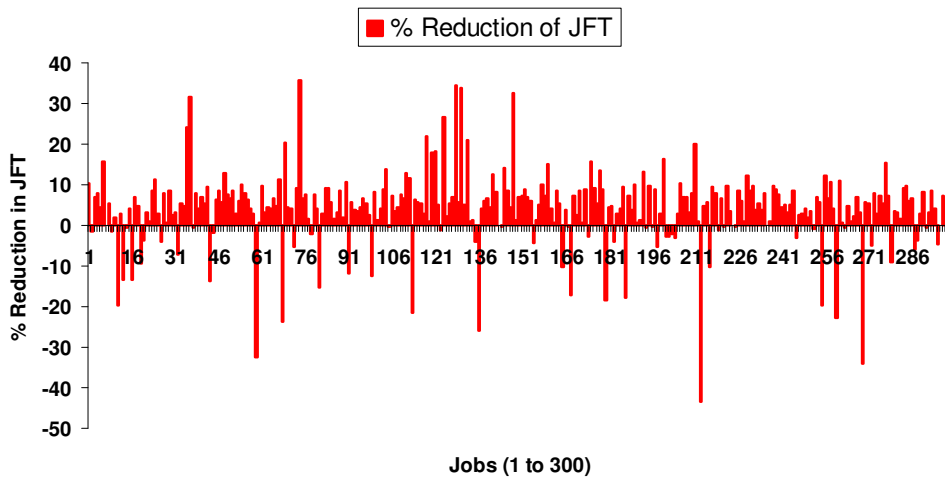


Figure 8.22: Percentage reduction in JFT using simulation based FAM

### 8.3 Testing reactive scheduling system: Parameters and tests

A series of tests were conducted after the development of the system. In the following sections, the test plans and parameters for the match-up rescheduling

system and the selective re-routing system are described. Most important results are shown, discussed and compared.

### 8.3.1 Testing the simulation assisted match-up rescheduling system

Table 8.15 shows the overview of tests conducted to check the effectiveness of the developed approach. The above tests are categorized into groups. In the first test group (test 1), the ASA is tested as it was implemented in steps and results are shown with varying processing times. The second test group (test 2) is also done similarly but considers fixed processing times similar to that found in some serial production environments. Before proceeding with the tests, a test is carried out to validate the Adaptation Synchrony Analysis (ASA) for a bigger system than the one described in chapter 5.

**Table 8.15 Test plan for match-up rescheduling system**

| Test number                    | To test the effect of  | On parameters   |
|--------------------------------|--|---|
| Test 1<br>• Test 1a, 1b and 1c | Processing time variations without ASA, limited ASA, and full ASA. | Reactive FAM and Starting time deviations, sequence deviations and make-span  |
| Test 2<br>• Test 2a and 2b     | Processing time variations with full ASA.                          | Reactive FAM and Starting time deviations, sequence deviations and make-span. |

#### 8.3.1.1 Validation of the detailed operation of the ASA

This section describes a test (using data in Table 8.16) that was performed in *eM-Plant* simulation software in order to check the functionality of the delayed method calls for shifting jobs within the implemented system. These delayed method calls are used with the match-up rescheduling algorithm in order to address the issues of adaption synchrony. The change charts shown in the implementation chapter are used to store the times needed to take jobs out of their predictively assigned buffer of a stage, transport it from there to another buffer and insert it there. The value of each of these change charts was kept in 5 minutes for this test. Therefore, if the executional exception occurred after 30 minutes, like in this test, the jobs should not be inserted into their newly assigned buffer before 15 minutes plus the estimated or required calculation time that will have elapsed within the simulation system after the occurrence of the exception. Some may even be inserted later, because the capacity of their new buffers was exhausted when they arrived and they therefore had to wait for a job or jobs to leave the buffer. Figure 8.23 shows the console output printing window of *eM-Plant* to which the executed job shifts during the calculation of the rescheduling solution are printed out. It can be seen that the required computation time using simulation was estimated to be

roughly 30 seconds.

**Table 8.16: Data for testing working of ASA**

| Nr. of jobs | Nr. of stages | Nr. of machines  | Processing times | Exception characteristics                         |
|-------------|---------------|------------------|------------------|---|
| 50          | 5             | At all stages: 5 | 5 to 45 minutes  | Occurred after 30 minutes, lasting for 90 minutes |

```

x Init ...
Rescheduling iteration 1...
Number of rescheduling runs: 5
Estimated calculation time for rescheduling solution: 30.7800
Job22 is moved from queue to Buffer3 at time: 1:00:00.0000
Reset ...
Init ...
Rescheduling iteration 2...
Job49 is moved to Buffer at time: 45:30.7800
Job22 is moved from queue to Buffer3 at time: 1:00:00.0000
Reset ...
Init ...
Rescheduling iteration 3...
Job22 is moved to Buffer at time: 45:30.7800
Job49 is moved to Buffer1 at time: 45:30.7800
Job19 is moved from queue to Buffer3 at time: 1:00:00.0000
Reset ...
Init ...
Rescheduling iteration 4...
Job19 is moved to Buffer at time: 45:30.7800
Job22 is moved to Buffer1 at time: 45:30.7800
Job49 is moved to Buffer4 at time: 45:30.7800
Job11 is moved from queue to Buffer3 at time: 1:00:00.0000
Reset ...
Init ...
Resimulation of the selected rescheduling iteration...
Job22 is moved to Buffer at time: 45:30.7800
Job49 is moved to Buffer1 at time: 45:30.7800
Job19 is moved from queue to Buffer3 at time: 1:00:00.0000
Reset ...
Init ...
Analysation of the selected rescheduling iteration with the FAM...
Job22 is moved to Buffer at time: 45:30.7800
Job49 is moved to Buffer1 at time: 45:30.7800
Job19 is moved from queue to Buffer3 at time: 1:00:00.0000
Reset ...
Init ...

```

**Figure 8.23: Output console window of *eM-Plant* simulation software**

This was estimated according to the principles explained in the ASA description in chapter 5. The jobs that were inserted into a buffer that had sufficient capacity for them, finished their shift at the simulation time of 45 minutes and roughly 30



seconds. Which is exactly the amount of 30 minutes (time point, when the exception occurred and the job shifting started) plus the estimated calculation time, plus the 15 minutes that are needed to physically shift the job to its newly assigned buffer. The console print out also shows, that buffer 3 was full when the job shifts were carried out. Therefore the jobs that were assigned to this buffer had to wait for capacity and finished their shift only after 30 minutes had elapsed from the time point of the exception. This shows, that the system correctly considers ASA and works according to its described method.

### 8.3.1.2 Test 1 with ASA implemented in steps, and with reactive FAM

#### 8.3.1.2.1 Test 1a without ASA and with reactive FAM

The first test is done without the ASA but with the FAM considering variation of job processing times on all stages. This implies that in the simulation model, as soon as there is an exception the reactive calculation starts immediately and reschedules jobs immediately (within the simulation model) without accounting for the fact that in the real world, these changes could take more time. The data for the first test is as follows:

**Table 8.17: Data for Test case 1**

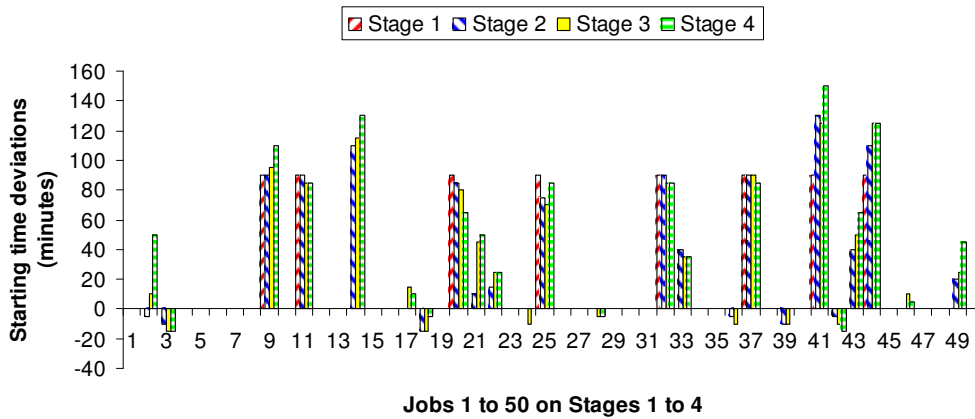
| Nr. of jobs | Nr. of stages | Nr. of machines   | Processing times              | Buffer capacity                   |
|-------------|---------------|-------------------|-------------------------------|-----------------------------------|
| 50          | 4             | At all Stages 1:4 | Varying from 15 to 25 minutes | Varying from 6 to 8 at each stage |

The exception duration was set to randomly occur for a duration of 40 minutes. The time required to obtain such information about the exception duration can be put into the change chart as explained in chapter 6. The aim of Test 1a was to test the effectiveness of the match-up rescheduling algorithm and the post rescheduling analysis using the simulation based FAM. Table 8.18 shows the overall results of the system. As seen in Table 8.18, the makespan of the best rescheduling iteration was close to that of the predictive schedule, while there was also significant reduction of the starting time deviations by this rescheduling iteration as compared the upper bound starting time deviations. The best rescheduling iteration rescheduled 4 jobs by itself. After post rescheduling analysis, it was seen that 1 job more was rescheduled by the FAM in order to solve bottleneck problems in the system which could occur in the future. In total out of the 50 jobs, only 5 were rescheduled if the user decided to pursue this post rescheduling analysis. Interestingly, using this post analysis the makespan was the same as the best rescheduling iteration, but resulted in very slight increase in

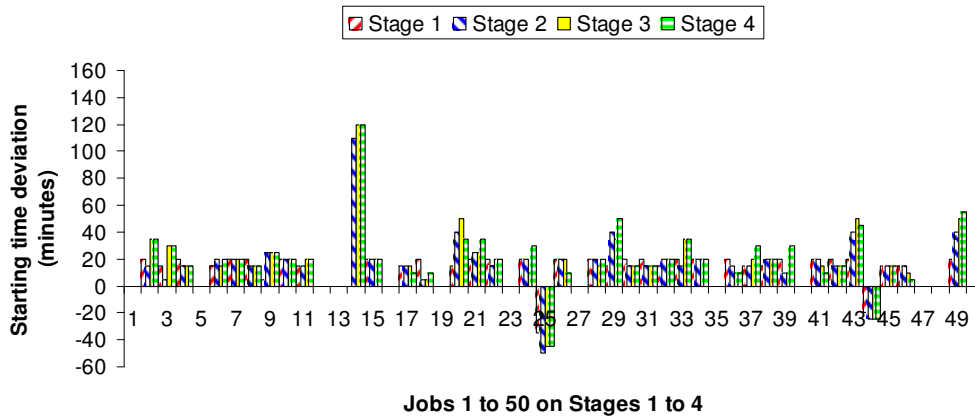
starting time deviations. Figures 8.24 and 8.25 show the per job starting time deviation reduction on each stage.

**Table 8.18: Summary of results for Test 1**

| Method                  | Make-span (hr:min.sec) | Starting time deviations (hr:min.sec) |
|-------------------------|------------------------|---------------------------------------|
| Best resched. iteration | 4:55.00                | 14.18                                 |
| Upper bound             | 5:45.00                | 19.12                                 |
| Predictive schedule     | 4:25.00                |                                       |
| Post resched. Analysis  | 4:55.00                | 14.28                                 |



**Figure 8.24: Comparison of upper bounds and predictive schedule**



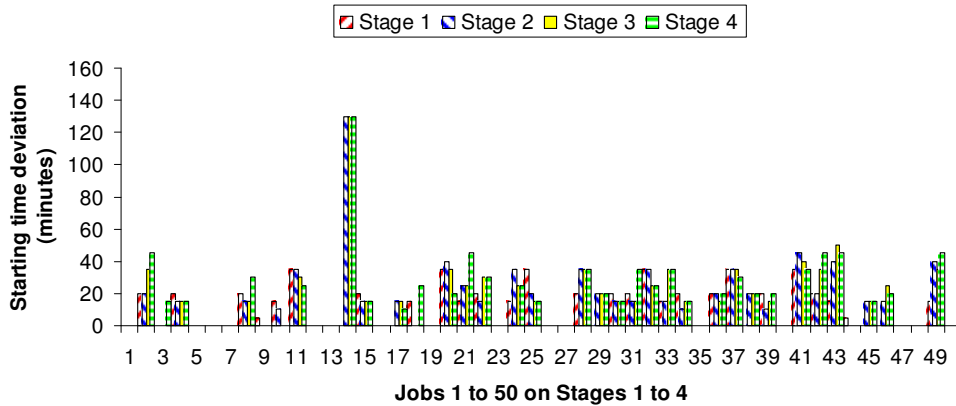
**Figure 8.25 Comparison of rescheduling result with predictive schedule**

It can be seen that as compared to the upper bound deviations, the

rescheduling system resulted in lower deviations, albeit not zero. In addition, since directly affected jobs are moved to other parallel machines, it increases the starting time deviations on that machine, though, in a small amount. Put together, it seems that the system does provide the user valuable inputs on what decisions he might take on system exceptions. He is given clear options with possibilities to select the rescheduling implementation scenario.

### 8.3.1.2.2 Test 1b with limited ASA but with reactive FAM

The aim of this test was to see the effect of gradually implementing the ASA into the system. Here an overall change time of 20 minutes was inserted into the system.



**Figure 8.26 Comparison of rescheduling results with predictive schedule**

Data for the third test was the same as that for the first test case. Here however, the ASA was implemented in this test to check its impact on the final result. Table 8.19 shows the results which need to be compared to Table 8.18 for the effects. As seen the starting time deviations and the make-span increased only slightly.

**Table 8.19: Summary of results for Test 1b**

| Method                  | Make-span (hr:min.sec) | Starting time deviations (hr:min.sec) |
|-------------------------|------------------------|---------------------------------------|
| Best resched. iteration | 5:05.00                | 15.33                                 |
| Upper bound             | 5:45.00                | 19.12                                 |
| Predictive schedule     | 4:25.00                |                                       |
| Post resched. Analysis  | 5:05.00                | 15.33                                 |

The reasons for this behaviour are described next. The most interesting result of using the ASA was that no jobs were selected for rescheduling using the FAM during post rescheduling analysis (as compared to test 1a – where 1 job was selected). Further, a different iteration was selected as a final rescheduling solution. The reason why this happened is because the ASA rescheduled the jobs at a later and appropriate point of time, thus, giving a rather realistic result – the result that by the time the changes would actually be ready on the shop floor, there would space to accommodate the changes in the buffers, thus needing no more rescheduling during the post rescheduling analysis phase. Figures 8.25 and 8.26 should be compared to starting time deviations between the test 1a and 1b. As seen, with the use of the ASA no significant differences in starting time deviations occur.

### 8.3.1.2.3 Test 1c with full ASA and with reactive FAM

In this test, a same configuration was kept as compared to test 1a and 1b, but with slightly increased exception duration. The ASA factors set-out time, transportation time, and set-in times discussed in chapter 6 were set to 5, 10 and 5 respectively. Table 8.20 show the data used while Table 8.21 shows the results obtained. Figure 8.27 shows the comparison of the Upper Bounds and the predictive plan. Figure 8.28 show the comparison of the rescheduling solution and the predictive plan.

Figure 8.28 when compared to Figure 8.25 and 8.26 shows that the deviations are quite similar meaning ultimately that with increasing ASA inclusion in the system, the system proves to be at least as effective. Hence it can be concluded that with higher processing time variations, the system effectively reduces deviations from its predictive trajectory when the ASA and the reactive FAM are considered.

**Table 8.20 Settings used for the test**

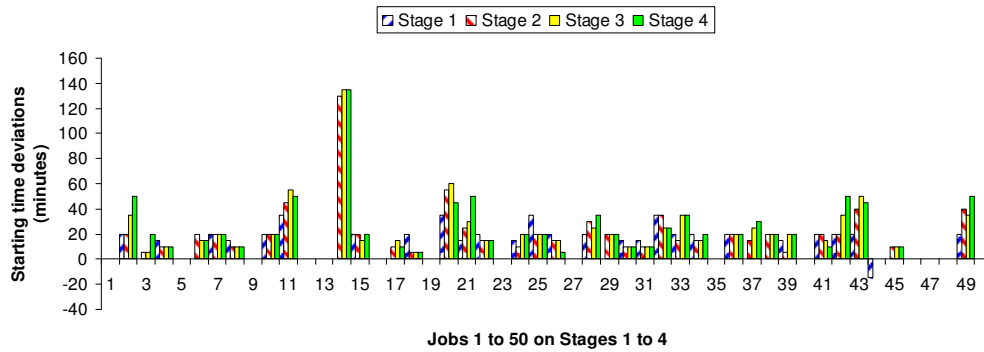
| <b>Variables</b>     | <b>Values</b>                        |
|----------------------|--------------------------------------|
| Processing times     | 15 to 25 random                      |
| Buffer sizes         | 8:7:7:8                              |
| Exception duration   | 1 hour 30 minutes                    |
| Exception occurrence | 30 minutes into the planning horizon |

**Table 8.21 Summary of results for test 1c**

| <b>Methods</b>              | <b>Makespan<br/>(hr:min.sec)</b> | <b>Starting time deviations<br/>(hr:min.sec)</b> |
|-----------------------------|----------------------------------|--|
| Predictive FAM              | 4:40                             |  |
| Upper Bounds                | 5:25                             | 18:04  |
| Best rescheduling iteration | 5:10                             | 15:33  |



**Figure 8.27: Comparison of upper bounds and predictive schedule**



**Figure 8.28: Comparison of rescheduling result with predictive schedule**

**8.3.1.3 Test 2 with ASA implemented in steps to test effectiveness of the rescheduling system**

**8.3.1.3.1 Test 2a without ASA but with reactive FAM**

A second test was carried out to test the influence of setting the processing times to be the same across all stages for all jobs, in order to check applicability for wider range of production systems, and increasing the exception duration to 1.5 hours. Table 8.22 shows the data used. The processing times were set to 20 minutes, 25 minutes, 20 minutes and 25 minutes respectively for stages 1 to 4.

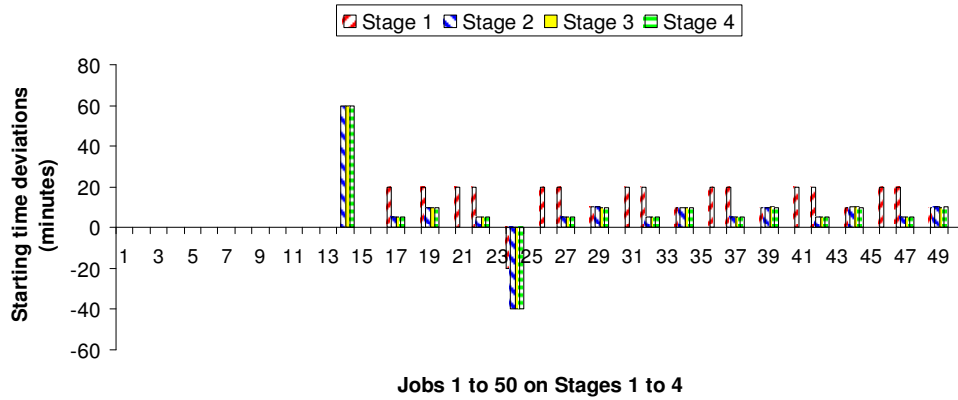
**Table 8.22: Data for Test case 2a**

| Nr. of jobs | Nr. of stages | Nr. of machines | Processing times | Buffer capacity                   |
|-------------|---------------|-----------------|------------------|-----------------------------------|
| 50          | 4             | Stage 1:4       | 20 Minutes       | Varying from 4 to 6 at each stage |
|             |               | Stage 2:4       | 25 Minutes       |                                   |
|             |               | Stage 3:4       | 20 Minutes       |                                   |
|             |               | Stage 4:4       | 25 Minutes       |                                   |

Table 8.23 shows the summary of the results obtained. It can be seen that even when the processing times were similar across stages, the system resulted in make-span and starting time deviations closer to the original predictive schedule.

**Table 8.23: Summary of results for Test 2a**

| Method                  | Make-span (hr:min.sec) | Starting time deviations (hr:min.sec) |
|-------------------------|------------------------|---------------------------------------|
| Best resched. iteration | 5:25.00                | 3.16                                  |
| Upper bound             | 5:55                   | 6.33                                  |
| Predictive schedule     | 5:15.00                |                                       |
| Post resched. Analysis  | 5:25.00                | 3.16                                  |



**Figure 8.29 Comparison of rescheduling result with the predictive schedule**



**Figure 8.30 Comparison of upper bounds and predictive schedule**

On the other hand, the upper bounds resulted in significant increase in make-span and starting time deviations. On conducting post rescheduling analysis, it was seen that the FAM did not reschedule any more jobs, and consequently the results after this analysis remained the same as that obtained by the best

rescheduling iteration. Figure 8.29 and 8.30 shows the graphical comparison between the rescheduling solution and the upper bound seen against the predictive schedule. Once again, it is seen that we could match-up the original schedule to the new one as much as possible.

### 8.3.1.3.2 Test 2b with ASA and with reactive FAM

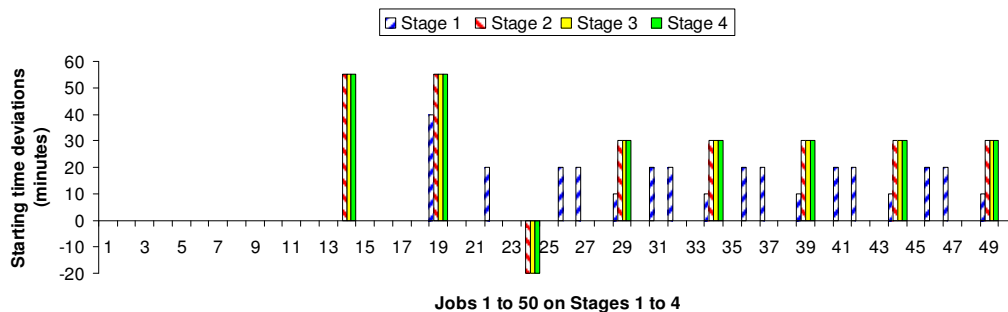
A second test was conducted within this group with similar processing times, and this time using the ASA with similar values as used in test group 1. Table 8.24 shows the data used while Table 8.25 shows the results obtained. Figure 8.31 shows the reduction in starting time deviations using the developed approach. Here it can be seen that although the makespan was better than the upper bounds, the starting time deviations were increased when compared to the result of Test 2a (or Figure 8.29). So it seems that with the ASA, when processing times are similar the system does not help in reducing deviations.

**Table 8.24 Data used for testing system**

|                       |                                  |
|-----------------------|----------------------------------|
| Variables             | Values                           |
| Processing times      | 20:25:20:25                      |
| Buffer sizes          | 8:7:7:8                          |
| Exception duration    | 50 minutes                       |
| Exception to occur at | 1 hour into the planning horizon |

**Table 8.25 Summary of results for Test 2b**

| Method                      | Makespan (hr:min.sec) | Starting time deviations (hr:min.sec) |
|-----------------------------|-----------------------|---------------------------------------|
| Predictive FAM              | 5:15                  |                                       |
| Upper Bounds                | 5:55                  | 6:33                                  |
| Best rescheduling iteration | 5:45                  | 5:09                                  |



**Figure 8.31 Comparison of rescheduling result with predictive schedule**

The system seems to work best for varying processing times. The reason for the worse results here is the fashion the jobs are rescheduled on machines with the maximum capacity. Jobs move in round robin fashion thus not providing the expected results.

### 8.3.2 Testing the selective re-routing system

Table 8.26 shows the test plan for testing the selective re-routing system. The main aims of all the tests here was to prove that it is possible to change a plan under execution as late as possible in the planning horizon, and as less as possible. Changing as late as possible was not possible using the match-up rescheduling system because the aim was to bring back the deviations to their original trajectories.

**Table 8.26 Test plan for testing selective re-routing system**

| Test number       | To test the effect of      | On parameters   |
|-------------------|----------------------------|---|
| Test 1 and Test 2 | Processing time variations | Reactive FAM, Make-span, Sequence deviations and number of jobs rescheduled |

These tests are designed randomly to test the effect of randomly set processing times among a set of jobs. Several tests more were conducted, but not shown here to limit the amount of data presented.

#### 8.3.2.1 Test 1 to check late change criteria using the selective re-routing system

In this test 50 jobs were taken, with a system consisting of 4 stages each with 4 machines. The processing time varied randomly between 5 to 30 minutes. The exception was set to occur after 30 minutes for a duration of 90 minutes. It can be seen from Table 8.27 that a saving of 1 hour can be expected if changes are made using the developed approach. Figure 8.32 shows the finishing times of the jobs using the methods developed and incorporating the time of the exception and the time of the change in the figure. It can be seen that the actual changes are made much later in the schedule. As seen the exception duration was set to about 35 % of the total make-span. Exceptions of more duration can be considered but in the author's opinion, such long exceptions can be better handled by other methods available to management to manage changes. The methods developed here are expected to work best for exception durations less than 30 % of the total make-span. This is because of the nature of the method of making changes from the last jobs of the stages.



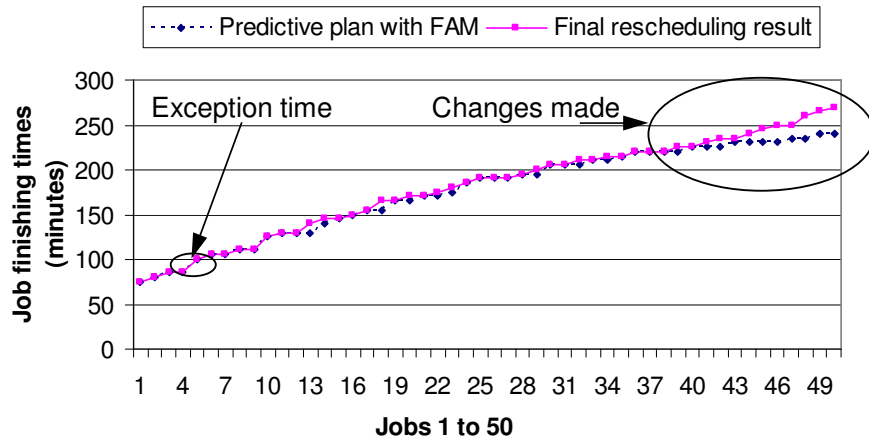


Figure 8.32 Job finishing times and events in the reactive system

Table 8.27 Summary of results for Test 1

| Method              | Makespan (hr:min.sec) | Number of jobs changed |
|---------------------|-----------------------|------------------------|
| Predictive FAM plan | 04:00.00              |                        |
| Upper bounds plan   | 05:30.00              |                        |
| Reactive FAM plan   | 04:30.00              | 6                      |

### 8.3.2.2 Test 2 to check late change criteria using the selective re-routing system

For this test the same settings were used, except for the processing times. The processing times were randomly set between 5 to 30 minutes for all jobs. 50 jobs were considered on 4 stages, each stage with 4 machines, and an exception of 90 minutes set to occur after 30 minutes of schedule progress.

Once again, as seen in Table 8.28, a reduction in makespan of 40 minutes occurred. Figure 8.33 shows the comparison of the job finishing times for the different methods and the start and end of the exception process and the change process.

Table 8.28 Summary of results for Test 2

| Method              | Makespan (hr:min.sec) | Number of jobs changed |
|---------------------|-----------------------|------------------------|
| Predictive FAM plan | 03:40.00              |                        |
| Upper bounds plan   | 05:05.00              |                        |
| Reactive FAM plan   | 04:15.00              | 6                      |

Here again, the changes took place very late in the schedule. The planners of the system have time in hand to make changes that could still result in a better make-span. Note that in this system, it was not required to use the Adaptation Synchrony Analysis (ASA) system as was developed for the match-up rescheduling system. Also since the jobs are the ones changed late into the planning horizon, it is not required to use the reactive FAM as was used in the match-up rescheduling system.

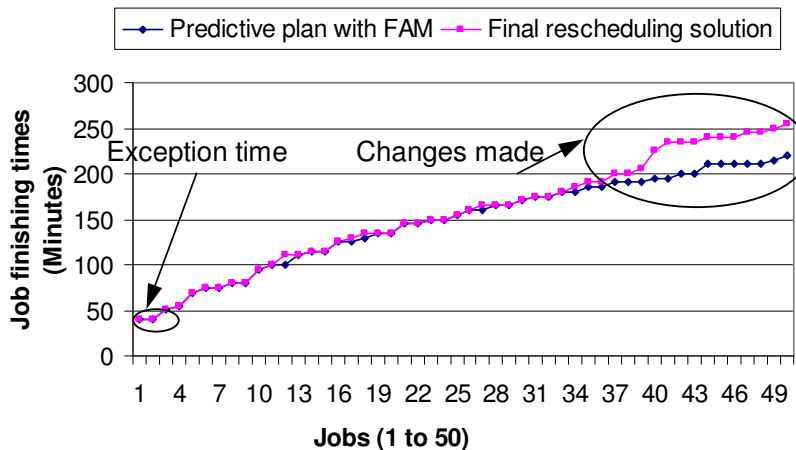


Figure 8.33 Job finishing times and events in the reactive system

## 8.4 Conclusions

In this chapter, we described the results of testing the systems developed in this thesis. The predictive system was thoroughly tested to result in definite quantitative benefits. Tests were carried out with parameters chosen systematically. A reason for this is the fact that the number of variables one could test the system for are prohibitively large. Testing the system on these tests give an impression about the workability of the system to different scenarios. As noticed in most cases, the rule-based simulation resulted in better performance measures than the pure optimization algorithm. Although this may not be the case in systems where several more elements are added, the system developed proves its applicability to such systems which can be further extended. The predictive system developed is unique in the way that it considered extreme details that can exist in production systems, and is yet able to schedule jobs based on specific constraints and delivery dates. The most important criteria of makespan were met. For the delivery time or lateness criteria, the system schedules job in such a way that it achieves the delivery times, and still as much as possible tries to achieve the

makespan criteria. The reactive system also resulted in several benefits. This system too was tested randomly on certain variables. As noticed, the system realistically considers the effect of including the actual change times on the overall change management solution. Specifically, we saw that the system rescheduled very few jobs and did tasks like post rescheduling analysis, which is also an important highlight of the system. Besides this the rescheduling system can also bring deviations back to their planning trajectory as much as possible. The approach in this thesis can be extended to consider other possibilities of rescheduling jobs like offloading work to suppliers, or compressing processing times of existing jobs by adding more resources and labor to the existing shop floor. The methods developed in this thesis have been developed as a decision support system, which can be extended further to include real-time information from a real production system and additional scheduling details and concerns.

# Chapter 9 Contributions, conclusions and further research

## 9.1 Contributions of this thesis

The specific contributions of this thesis to the state of the art research in scheduling and rescheduling are as follows:

1. A system was developed which resulted in knowledge on how a combination of simulation and optimization can be realized to result in additional benefits for scheduling a flexible production system configuration. This system resulted in definite improvements in system performance for the flexible production system as seen in results. Specific features about this system are:
  - 1.1 The optimization algorithm which extends and builds on top of an established algorithm is relatively trouble-free, yet produces fairly good results in terms of make-span and delivery time optimization.
  - 1.2 The optimization system considers special and standard job flows with or without due dates and with or without specific part flows within the same system alongwith constraints like limited buffer spaces, materials, tools and resource availability which makes the problem more complex and which none of the researchers have considered in their research. The result of this consideration is that the system becomes applicable to a wider range of manufacturing situations where part flow depends on technological (machining) restrictions imposed by the parts and the machines. A system which works has been developed as a starting point, and this system can be expanded to include more details in the future, resulting in a comprehensive planning and scheduling tool within eM-Plant.
  - 1.3 A first small step of combining simulation and optimization using *eM-Plant* simulation software was taken in this thesis, which can be extended in the future. *eM-Plant* simulation software already offers the most powerful modeling environment with great flexibility in modeling complex situations. The rule based simulation system can be applied in a general way to encompass the broader manufacturing organization elements and events. The rules developed in this system were developed for buffer elements and to handle bottleneck and optimality problems. Similarly, more general rules for other elements like transportation equipment, pallets, etc or

any other events, can be written in the form of these rules. The principles developed here are easy to apply, with the possibility to set, edit and write own rule generator logic for these elements to suit custom situations in the simulation. This way, complexity is handled, as well as ensuring that the final scheduling solution is as near to the optimal one or atleast a good one, which encompasses great details.

2. A system was developed for rescheduling which resulted in knowledge on how and if a combination of simulation and optimization can result in additional benefits. This system resulted in improvements in system performance for the flexible production system as seen in results. Specific features and contrubutions of this system are:

2.1 The rescheduling system provided drastic reduction in solutions for rescheduling, whilst also considering and solving problems of adaptation synchrony analysis.

2.2.1 This was done by the optimization algorithm, which provided the simulation system with alternative solutions, alongwith providing feedback on ASA.

2.2.2 The simulation system did not simulate each and every constellation consisting of every job, but the overall iteration, by using the updating capacity principle.

2.2 The rule based Flow Analyser Module (FAM) simulation system was used to provide problem free execution of the rescheduling solution. It was seen that future problems can indeed be detected, and solved in the current rescheduling step. The number of jobs rescheduled were kept to a bare minimum by using the rule based FAM system, thus meaning, that the entire schedule need not be rescheduled, but only a few jobs further, for which problems are expected.

2.3 Both the above points were considered, whilst bringing back deviation to the planned trajectory as much as possible.

2.4 The selective re-routing system changed the schedule as late as possible in the planning horizon. It was seen that good results were obtained here too. The planners can neglect the exceptions, and yet be able to account for them at a later time point.

Results show that these approaches indeed provide critical information and helps in improving the efficiency of the schedule before and after implementation in the real world by considering known and unknown events and exceptions.

## 9.2 Conclusions

The predictive system has several practical applications. Many high volume production facilities have several separate flow shops. The process in such facilities is such that machines are flexible or interchangeable at each stage and therefore practically similar. Some production facilities also have special expertise in machining a family of parts, where each part follows the same sequence, but each machine is flexible to accommodate the slight variety in parts. Assembly lines, in which more than one type of product may be manufactured and each work station has multiple machines, is also an obvious application of this problem. Similarly, the situation where a parallel machine is added to ease pressure on a bottleneck facility, and/or to increase production capacities can be viewed as an application of the suggested problem. The reactive system could be used in situations where Just-In-Time or Just-In-Sequence production methods are used, where parts and supplies are delivered according to a precalculated plan, and deviations in production are not advisable.

In this thesis, several test examples were taken, solved and simulated to provide a greater understanding of the underlying system. The solutions generated alternative policies in both the predictive and reactive areas, are in fact the significant processes which need to be handled in the predictive and reactive planning phases. Consequently, this means that significant processes can in fact be automatically detected, and handled efficiently using a combination of optimization algorithms and simulation.

### **9.3 Further research**

During the course of this research work, the following observations were made which can be addressed by further research:

1. In this work, scheduling of special jobs with routes only was limited to scheduling them last. These jobs could be clubbed with the standard jobs and a procedure could be developed, where such special jobs are scheduled according to the critical stage calculation. However, this may or may not yield better make-span, because of the fact that the selected special job may have co-incidently a routing on the latest available machines on all the stages. A method can be developed and tested to see if combining standard jobs with such special jobs and using the critical stage calculation as a guiding method, is possible and helpful at all.
2. In this work, scheduling standard jobs was done by considering the earliest starting times, processing times and the tails of special jobs in the critical stage calculation. Though this method gave good results, the starting times of special jobs were not updated when calculating the critical stage. This was because,

the critical stage calculation was directed at the standard jobs, and not the special jobs which are not going to be scheduled anyway. Further research in this direction could be considering actual and updated starting times of the special jobs. This may lead improvement over the method used currently in this work.

3. In this work, rescheduling was done by considering all jobs – standard and special on the same priority level. Special jobs with routings, were not considered. This was done to first prove that the rescheduling methods developed here work and are beneficial. Including jobs with special routing can be included in the future.
4. The selective rerouting system did not use the reactive FAM and the ASA methods because jobs which are last in the planning horizon are rescheduled last, giving ample time to make changes in the real world. The reactive FAM and the ASA can be implemented for the selective rerouting algorithm, though which will be helpful only when the exception occurs close to the end of the planning horizon.
5. One area where further improvement is possible is the detailing of the simulation model. For example, until now the PMFS model used in the system does not contain any transportation elements like forklifts or containers. Other examples include the modelling of workers, set-up times, or even very specific events etc. In order to make the model more detailed, these elements should be integrated into the model using the developed base of the Flow Analyser Module (FAM). *eM-Plant* already models these elements as standard modelling objects within its object library. This makes a more detailed schedule analysis possible.
6. The effects and costs of rescheduling a current production plan on other nodes of the bigger manufacturing organization and the supply chain could be developed. This could be of special interest to industry and research.
7. In the match-up rescheduling area, it seems that the schedule cannot be fully brought back to its predictive level (though, better results are obtained against doing no rescheduling at all) due to the limited possibilities considered at the shop floor. Other possibilities for reduction of starting time deviations could be encompassing the wider manufacturing system, and alternative possibilities such as compressing processing times of jobs by adding more resources and manpower to the already present equipment. These other possibilities will have impact on costs, and this might prove an additional topic of further research.
8. The rescheduling system can be developed to work with probabilistic exception durations, where the best and worst case exception durations can be taken to develop estimated on rescheduling efforts required. In cases where exception durations are hard to estimate, over estimating the duration slightly by making

approximations, can be tried to be used to estimate rescheduling efforts. Nevertheless, in such cases, the Adaptation Synchrony Analysis, and the reactive FAM should still work as developed.



## VI ACKNOWLEDGEMENT

I express heartfelt gratitude towards Professor Wilhelm Dangelmaier for his mentoring and support during the time I spent at his group. Without his direction, this work simply wouldn't have been completed. I also extend my thanks to the International Graduate School Dynamic Intelligent Systems at Paderborn University for extending me their support.

Looking back to the three years that I spent in Paderborn, I see the everyday difficulties, small but important things to do everywhere, at work, at home. Sometimes, it was depressing; the mood was down over uncertainty of my future. Sometimes i was upbeat as I got results in my work. Nevertheless, I could find my way as time went by. This was possible only because several people offered me their assistance in every phase of my work and stay in Paderborn. I truly appreciate this assistance, and I find myself extremely lucky to have been close to such wonderful people. Besides earning my PhD, I could also learn important lessons of discipline, punctuality and the german way of doing things in a systematic way. I consider these soft skills particularly necessary in an increasing global work arena.

Finally I want to thank all my colleagues for their everyday assistance in small and big problems. Particularly, I want to thank Dr. Bengt Mueck for being patient to listen to my ideas and provide constructive criticism. He particularly got me close to the German way of life by telling me a lot of stories right from german soccer to the german military! I also want to thank Dr. Werner Franke for all his warmth that he extended to me during my stay in Paderborn. I thank Benjamin Klöpper with whom I always had useful discussions. I also thank Melanie Fearn for always assisting me in office work. In particular I thank my student Thomas Seeger very much for assisting me in the implementation of some difficult algorithms. Finally I want to thank Prof. Suhl and Prof. Meyer auf der Heide for reviewing this piece of work. I also want to express my gratitude to them for giving me this once in a lifetime opportunity to work with them. I think the combination of business computing and manufacturing competence will help me boost my career in the future. I wish to thank Prof. Wilhelm Schäfer and Dr. Eckhard Steffen at the International Graduate School of Dynamic Intelligent Systems for giving me an opportunity of a life time to undertake this studies.

I know I will miss Paderborn and all these people very much, and it's never easy to leave a place you liked so much...

## VII REFERENCES

- [1] Abumaizar, R. J., and J. A. Svestka. 1997. Rescheduling job shops under random disruptions. *International Journal of Production Research* 35 (7): 2065-2082.
- [2] Adams, J., Balas, E., Zawack, D., 1988, The shifting bottleneck procedure for job shop scheduling, *Management science*, Vol. 34, No.3, pp. 391-401.
- [3] Akturk, M. S., and E. Gorgulu. 1999. Match-up scheduling under a machine breakdown. *European Journal of Operational Research* 112 (1): 81-97.
- [4] Baptiste, P., Favrel, J., 1993. Taking into account the rescheduling problem during the scheduling phase. *Production planning and control* 4, 349-360.
- [5] Baker, K.R., *Introduction to sequencing and scheduling*, John-Wiley and sons, Inc, 1974, ISBN: 0-471-04555-1.
- [6] Banks, J., *Handbook of simulation – Principles, Methodology, Advanced, Applications, and Practice*. John-Wiley and sons, Inc, 1998, ISBN 0-471-13403-1.
- [7] Bhaskaran, K., Pinedo; M., 1991. Dispatching. In: Salvendy, G. (Ed.), *Handbook of Industrial Engineering*. John Wiley, New York, Chapter 83.
- [8] Billaut, J.C., Roubellat, F., 1996. A new method for workshop real time scheduling. *International journal of production research* 34, 1555-1579.
- [9] Bock, S., *Lecture on Real-Time control of complex production and logistic processes*, Summer semester 2005, Paderborn University.
- [10] Brah, S. A, Hunsucker, J. L., 1991, Branch and Bound algorithm for the flow shop with multiple processors, *European Journal of Operational Research*, Vol. 51, pp- 88-99.
- [11] Brockmann, K., Dangelmaier, W., 1997. Ein paralleler Branch and Bound – Algorithmus zur Minimierung der Zykluszeit in Fleißlinien mit parallelen maschinen, Report of Sonderforschungsbereich 376, Massive Parallelität – Algorithmen entwurfsmethoden anwendungen, Universität – Gesamthochschule Paderborn, Germany.
- [12] Carlier, J., 1987, Scheduling jobs with release dates and tails on identical machines to minimize the make span, *European Journal of Operational Research*, Vol. 29, pp. 298-306.
- [13] Carlier, J., 1982, The one-machine sequencing problem, *European Journal of Operational Research*, Vol. 11, pp. 42-47.
- [14] Chen, F., Drezner, Z., Ryan, J.K., Simchi-Levi, D., 2000. Quantifying the bullwhip effect in a simple supply chain: The impact of forecasting, lead-times and information. *Management Science* 46, 436-443.
- [15] Cheng, J., Karuno, Y., and Kise, H. 2001. A shifting bottleneck approach for a parallel machine flow shop scheduling problem. *Journal of Operations*

- Research, Society of Japan, Vol. 44, No. 2, pp 140 – 156.
- [16] Chong, C. S., A. I. Sivakumar, and R. Gay. 2003. Simulation-based scheduling for dynamic discrete manufacturing. In Proceedings of the 2003 Winter Simulation Conference, ed. S. Chick, P. J. Sanchez, D. Ferrin, and D. J. Morrice, 1465 - 1473. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.
  - [17] Church, L.K., Uzsoy, R., 1992. Analysis of periodic and event driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing* 5, 153-163.
  - [18] Dangelmaier, Wilhelm; Becker, B. D., SIMPLE++ : A new object oriented simulation system with expert system support. The Eastern Simulation Conference (ESC), Orlando, Florida, 18 – 21 April 1988.
  - [19] Dutta, A., 1990. Reacting to scheduling exceptions in FMS environments. *IIE Transactions* 22, 300-314.
  - [20] Drake, G., Smith, J., Peters, B., 1995. Simulation as a planning and scheduling tool for flexible manufacturing systems. In proceedings of the 1995 Winter Simulation Conference, ed. Alexopoulos, C., Kang, K., Lilegdon, W., Goldsman, D. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.
  - [21] Gupta, J.N.D., Ruiz-Torres, A.J., 2000, Minimizing makespan subject to minimum total flow time on identical parallel machines, *European Journal of Operational Research*, Vol. 125, No.5, pp.616-625.
  - [22] Haldun, A., Lawley, M., McKay, K., Mohan, S., Uzsoy, R., 2005. Executing production schedules in the face of disturbances: A review. *European Journal of Operational Research* 161 (2005), 86-110.
  - [23] Harmonosky, C. M., R. H. Farr, and M. C. Ni. 1997. Selective rerouting using simulated steady state system data. In Proceedings of the 1997 Winter Simulation Conference, ed. S. Andradottir, K. J. Healy, D. H. Withers, and B. L. Nelson, 1293 - 1298. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.
  - [24] Hutchison, J., Leong, K., Snyder, D., Ward, P., 1991. Scheduling approached for random job shop flexible manufacturing systems. *International Journal of Production Research* 29, 1053-1067.
  - [25] Jain, A. K., and H. A. Elmaraghy. 1997. Production scheduling/rescheduling in flexible manufacturing. *International Journal of Production Research* 35 (1): 281-309.
  - [26] Kouvelis, P., Yu, G., 1997. *Robust Discrete Optimization and its Applications*. Kluwer Academic Publishers.
  - [27] Krauth, J. 2005. Simulation based Production Planning and Scheduling: Sim-Serv's experience. Technical Notes, *Simulation News Europe*. Issue

43. July 2005. ISSN 0929-2268.

- [28] Lawrence, S. R., Sewell, S. E., 1997. Heuristic, optimal, static and dynamic schedules when processing times are uncertain. *Journal of Operations Management* 15, 71-82.
- [29] Leon, V. J., S. D. Wu, and R. H. Storer. 1994. A game-theoretic control approach for job shops in the presence of disruptions. *International journal of production research* 32: 1451-1476.
- [30] Leon, V. J., S. D. Wu, and R. H. Storer. 1993. Robustness measures and robust scheduling for job shops. *IIE Transactions* 26, 32-43.
- [31] Lin, G. Y. J., Solberg, J. J., 1992. Integrated shop-floor control using autonomous agents. *IIE Transactions* 24, 57-71.
- [32] Manivannan, S., Banks, J., 1991. Real-Time Control of a manufacturing cell using knowledge-based simulation. In *Proceedings of the 1991 Winter Simulation Conference*, ed. B. Nelson, W. D. Kelton, G. M. Clark, 251 - 260. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.
- [33] Matsuura, H., Tsubone, H., Kanezashi, M., 1993. Sequencing, dispatching, and switching in a dynamic manufacturing environment. *International Journal of Production Research* 31, 1671-1688.
- [34] McKay, K. N., Safayeni, F., Buzacott, J. A., 1995. An information systems based paradigm for decision making in rapidly changing industries. *Control Engineering Practice* 3 (1), 77-88.
- [35] Mehta, S. V., Uzsoy, R., 1999. Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer-Integrated Manufacturing* 12, 15-38.
- [36] Mueck, B., Dangelmaier, W. and Fischer, M. 2003. Components for the active support of the analysis of material flow simulations in a virtual environment, In *Proceedings of the 15th European Simulation Symposium*, ed. A. Verbraeck and V. Hlupic, 367-371. Erlangen Germany: SCS publishing house e.V.
- [37] Musselman, K., Uzsoy, R., 2001. Advanced planning and scheduling for manufacturing. In: Salvendy, G. (Ed.), *Handbook of Industrial Engineering*. John Wiley.
- [38] Musselman, K., Reilly, J., Duket, S., 2002. The role of simulation in advanced planning and scheduling. In *proceedings of the 2002 Winter Simulation Conference*, ed. Yücesan, E., Chen, C., Snowdon, J., and Chranes, J. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.
- [39] O'Kane, J. F., 2000. A knowledge-based system for reactive scheduling decision-making in FMS. *Journal of Intelligent Manufacturing* 11 (5), 461-474.

- [40] Phadnis, S., Irani, S., 2003, Development of a new heuristic for scheduling flow-shops with parallel machines by prioritizing bottleneck stages, Transactions of the SDPS, Vol. 7, No. 1, pp 87-97.
- [41] Pinedo, M., Scheduling Theory, Algorithms, and systems, second edition, Prentice Hall, 2001.
- [42] Russel, S., Norvig, P., Artificial intelligence – A modern approach, Prentice Hall, 2003, ISBN 0-13-080302-02.
- [43] Sun, D., Lin, L., 1994. A dynamic job shop scheduling framework: A backward approach. International Journal of Production Research 32, 967-985.
- [44] Shafaei, R., Brunn, P., 1999a. Workshop scheduling using practical (innaccurate) data. Part 1. The performance of heuristic scheduling rules in a dynamic job shop environment using a rolling horizon approach. International Journal of Production Research 37, 3913-3925.
- [45] Tremble, J., Hunter, K., Concannon, K., 2003. Simul8-Planner: Simulation based planning and scheduling. In Proceedings of the 2003 Winter Simulation Conference, ed. S. Chick, P. J. Sanchez, D. Ferrin, and D. J. Morris. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.
- [46] Technomatix *eM-Plant* user guide 2007.
- [47] Ten Hompel, M., Heidenblut, V., 2005. Taschenlexikon Logistik – Abkürzungen, Definitionen und Erläuterungen der wichtigsten Begriffe aus Materialfluss und Logistik, VDI series, ISBN3-540-28581-4.
- [48] Wan, Y.-W., 1995. Which is better, off-line or real-time scheduling? International journal of Production Research 33, 2053-2059.
- [49] Wu, S. D., Wysk, R. A., 1989. An application of discrete-event simulation to on-line control and scheduling of flexible manufacturing. International Journal of Production Research 27 (9).
- [50] Wu, S. D., Byeon, E., Storer, R. H., 1999. A graph-theoretic decomposition of job shop scheduling problems to achieve scheduling robustness. Operations Research 47, 113-124.
- [51] Yamamoto, M., Nof, S. Y., 1985. Scheduling\re-scheduling in a manufacturing operation system environment. International Journal of Production Research 23, 795-722.
- [52] Vollman, T., Berry, W., Whybark, D., Jacobs, F., 2005. Manufacturing planning and control systems for supply chain management. Mc-Graw Hill, ISBN 007144033X.
- [53] [www.thefreedictionary.com](http://www.thefreedictionary.com)

## VIII APPENDICES

### APPENDIX 1: Detail iterations for predictive scheduling algorithm and validation

#### Iteration 2

At the beginning of this iteration, job 3 is already in the list of scheduled jobs. In this iteration it is checked if special jobs (especially job 5 and 7) can be scheduled since they have delivery times. The check fails in step 11, and hence the iteration proceeds with scheduling other standard jobs. Stage 1 is the critical stage and job 1, 2, and 4 have common tails on the critical stage. Hence, maximum processing times for these jobs is checked. Then it is found that job 4 has the maximum processing time on the critical stage and is hence selected to be scheduled on earliest available machines on stage 1 and stage 2. The machine ready times are updated and shown in Table A7. Note here that during the calculation of the critical stage, special jobs are considered to act as standard jobs when considering their starting times. This is logical since, we know that special jobs are not going to be scheduled anyway. The idea is to only include the contribution of special jobs to the overall calculation of the critical stage, and then select a particular standard job.

**Table A1 Job ready times at stage - 1 ( $a_{i,1}$ )**

|           | Jobs |   |   |              |            |   |   |   |    |  |
|-----------|------|---|---|--------------|------------|---|---|---|----|--|
|           | 1    | 2 | 4 | 5            | 6          | 7 | 8 | 9 | 10 |  |
| $a_{i,1}$ | 0    | 0 | 0 | 1:00:00.0000 | 30:00.0000 | 0 | 0 | 0 | 0  |  |

**Table A2 Tails for each job at each stage**

| Jobs | Tails                  |                        |
|------|------------------------|------------------------|
|      | at Stage 1 - $q_{i,j}$ | at Stage 2 - $q_{i,j}$ |
| 1    | 20                     | 0                      |
| 2    | 20                     | 0                      |
| 4    | 20                     | 0                      |
| 5    | 15                     | 0                      |
| 6    | 20                     | 0                      |
| 7    | 20                     | 0                      |
| 8    | 30                     | 0                      |
| 9    | 15                     | 0                      |
| 10   | 15                     | 0                      |

**Table A3 Earliest start times at stage 1 for all jobs**

| <b>Jobs</b> | <b><math>s_{i,1} \rightarrow</math> Stage 1</b> |
|-------------|---|
| 1           | 0   |
| 2           | 0   |
| 3           | 0   |
| 4           | 0   |
| 5           | 1:00:00.0000                                    |
| 6           | 30:00.0000                                      |
| 7           | 0   |
| 8           | 0   |
| 9           | 0   |
| 10          | 0   |

**Table A4 Earliest start times at stage 2 for all jobs**

| <b>Jobs</b> | <b><math>s_{i,2} \rightarrow</math> Stage 2</b> |
|-------------|---|
| 1           | 25  |
| 2           | 25  |
| 3           | 25  |
| 4           | 30  |
| 5           | 75  |
| 6           | 60  |
| 7           | 10  |
| 8           | 15  |
| 9           | 30  |
| 10          | 15  |

**Table A5 Lower bounds computed for both stages**

| <b>Stage 1</b>       |                      | <b>Stage 2</b>       |                      |
|----------------------|----------------------|----------------------|----------------------|
| <b>Lower Bound 1</b> | <b>Lower Bound 2</b> | <b>Lower Bound 1</b> | <b>Lower Bound 2</b> |
| 45                   | <b>112.5</b>         | 45                   | <b>100</b>           |
| 45                   |                      | 45                   |                      |
| 50                   |                      | 50                   |                      |
| <b>90</b>            |                      | <b>90</b>            |                      |
| 80                   |                      | 80                   |                      |
| 30                   |                      | 30                   |                      |
| 45                   |                      | 45                   |                      |
| 45                   |                      | 45                   |                      |
| 45                   |                      | 30                   |                      |

**Table A6 Jobs scheduled list after this iteration**

|       |
|-------|
| Job 3 |
| Job 4 |

**Table A7 Updating machine ready times on stage 1 and stage 2 after iteration 2**

| Stage 1   |           | Stage 2   |           |
|-----------|-----------|-----------|-----------|
| Machine 1 | Machine 2 | Machine 1 | Machine 2 |
| 25        | 30        | 50        | 50        |

**Iteration 3**

At the beginning of this iteration, job 3 and 4 is already in the list of scheduled jobs. In this iteration it is checked if special jobs (especially job 5 and 7) can be scheduled since they have delivery times. The check fails in step 11, and hence the iteration proceeds with scheduling other standard jobs. In the third iteration, the critical stage is stage 2. Since both jobs 1 and 2 have the same tails on the critical stage, the maximum processing times are checked. Since they are also same, the tie is broken by selecting a job with a lower number. So job 1 is selected for scheduling and it is scheduled on the earliest available machines namely on machine 1 on stage 1 and on machine 1 on stage 2. The machine ready times are updated as shown in Table A14. Note the updated earliest starting times obtained from the job ready times and the updated machine ready times on both stages.

**Table A8 Job ready times at stage - 1 ( $a_{i,1}$ )**

|           | Jobs |   |              |            |   |   |   |    |
|-----------|------|---|--------------|------------|---|---|---|----|
|           | 1    | 2 | 5            | 6          | 7 | 8 | 9 | 10 |
| $a_{i,1}$ | 0    | 0 | 1:00:00.0000 | 30:00.0000 | 0 | 0 | 0 | 0  |

**Table A9 Tails for each job at each stage**

| Jobs | Tails                  |                        |
|------|------------------------|------------------------|
|      | at Stage 1 - $q_{i,j}$ | at Stage 2 - $q_{i,j}$ |
| 1    | 20                     | 0                      |
| 2    | 20                     | 0                      |
| 5    | 15                     | 0                      |
| 6    | 20                     | 0                      |
| 7    | 20                     | 0                      |
| 8    | 30                     | 0                      |
| 9    | 15                     | 0                      |
| 10   | 15                     | 0                      |



**Table A10 Earliest start times at stage 1 for all jobs**

| <b>Jobs</b> | <b><math>s_{i,1} \rightarrow</math> Stage 1</b> |
|-------------|---|
| 1           | 25  |
| 2           | 25  |
| 5           | 1:00:00.000                                     |
| 6           | 30:00.0000                                      |
| 7           | 25  |
| 8           | 25  |
| 9           | 25  |
| 10          | 25  |

**Table A11 Earliest start times at stage 2 for all jobs**

| <b>Jobs</b> | <b><math>s_{i,2} \rightarrow</math> Stage 2</b> |
|-------------|---|
| 1           | 50  |
| 2           | 50  |
| 5           | 75  |
| 6           | 50  |
| 7           | 50  |
| 8           | 50  |
| 9           | 55  |
| 10          | 50  |

**Table A12 Lower bounds computed for both stages**

| <b>Stage 1</b>       |                      | <b>Stage 2</b>       |                      |
|----------------------|----------------------|----------------------|----------------------|
| <b>Lower Bound 1</b> | <b>Lower Bound 2</b> | <b>Lower Bound 1</b> | <b>Lower Bound 2</b> |
| 70                   | <b>122.5</b>         | 70                   | <b>127.5</b>         |
| 70                   |                      | 70                   |                      |
| <b>90</b>            |                      | <b>90</b>            |                      |
| 85                   |                      | 70                   |                      |
| 55                   |                      | 70                   |                      |
| 70                   |                      | 80                   |                      |
| 55                   |                      | 70                   |                      |
| 55                   |                      | 65                   |                      |

**Table A13 Jobs scheduled list after this iteration**

|       |
|-------|
| Job 3 |
| Job 4 |
| Job 1 |

**Table A14 Updating machine ready times on stage 1 and stage 2 after iteration 3**

| Stage 1   |           | Stage 2   |           |
|-----------|-----------|-----------|-----------|
| Machine 1 | Machine 2 | Machine 1 | Machine 2 |
| 50        | 30        | 70        | 50        |

**Iteration 4**

At the beginning of this iteration, job 3, 4 and 1 is already in the list of scheduled jobs. In this iteration it is checked if special jobs (especially job 5 and 7) can be scheduled since they have delivery times. The check fails in step 11, and hence the iteration proceeds with scheduling other standard jobs. In this iteration, the critical stage was stage 2 because of the highest lower bounds, and from the available jobs, job 2 is selected and scheduled earliest on machine 2 on stage 1 and on machine 2 on stage 2. Note that to schedule jobs earliest, they also have to be available at that time. So the maximum of the job ready times and the machine ready times are taken on stage 1 to schedule a particular job earliest. The machine ready times are then updated. Refer to the table A17 on the updated machine ready times after this iteration.

**Table A15 Job ready times at stage - 1 ( $a_{i,1}$ )**

|           | Jobs |              |            |   |   |   |    |
|-----------|------|--------------|------------|---|---|---|----|
|           | 2    | 5            | 6          | 7 | 8 | 9 | 10 |
| $a_{i,1}$ | 0    | 1:00:00.0000 | 30:00.0000 | 0 | 0 | 0 | 0  |

**Table A16 Tails for each job at each stage**

| Jobs | Tails                  |                        |
|------|------------------------|------------------------|
|      | at Stage 1 - $q_{i,j}$ | at Stage 2 - $q_{i,j}$ |
| 2    | 20                     | 0                      |
| 5    | 15                     | 0                      |
| 6    | 20                     | 0                      |
| 7    | 20                     | 0                      |
| 8    | 30                     | 0                      |
| 9    | 15                     | 0                      |
| 10   | 15                     | 0                      |

**Table A17 Updating machine ready times on stage 1 and stage 2 after iteration 4**

| Stage 1   |           | Stage 2   |           |
|-----------|-----------|-----------|-----------|
| Machine 1 | Machine 2 | Machine 1 | Machine 2 |
| 50        | 55        | 70        | 75        |

**Table A18 Earliest start times at stage 1 for all jobs**

| Jobs | $s_{i,1} \rightarrow$ Stage 1 |
|------|-------------------------------|
| 2    | 30                            |
| 5    | 1:00:00.000                   |
| 6    | 30:00.0000                    |
| 7    | 30                            |
| 8    | 30                            |
| 9    | 30                            |
| 10   | 30                            |

**Table A19 Earliest start times at stage 2 for all jobs**

| Jobs | $s_{i,2} \rightarrow$ Stage 2 |
|------|-------------------------------|
| 2    | 55                            |
| 5    | 75                            |
| 6    | 60                            |
| 7    | 50                            |
| 8    | 50                            |
| 9    | 60                            |
| 10   | 50                            |

**Table A20 Lower bounds computed for both stages**

| Stage 1       |               | Stage 2       |               |
|---------------|---------------|---------------|---------------|
| Lower Bound 1 | Lower Bound 2 | Lower Bound 1 | Lower Bound 2 |
| 75            | 115           | 75            | 117.5         |
| <b>90</b>     |               | <b>90</b>     |               |
| 80            |               | 80            |               |
| 60            |               | 70            |               |
| 75            |               | 80            |               |
| 75            |               | 75            |               |
| 60            |               | 65            |               |

**Table A21 Jobs scheduled list after this iteration**

|       |
|-------|
| Job 3 |
| Job 4 |
| Job 1 |

Job 2

**Iteration 5**

At the beginning of this iteration, job 3, 4, 1 and 2 are already in the list of scheduled jobs. In this iteration, only special jobs 5, 6, 7, 8, 9 and 10 are left, and standard jobs are already scheduled. The special job scheduling system works by first looking at jobs with delivery time and routing and then jobs with only delivery times, followed by jobs with only routing. The jobs with only routings (job 6, 8, 9 and 10 in this case) is considered for scheduling last. So in this iteration, job 7 is selected to checked to see if meets the scheduling criteria of delivery times. This calculation is performed by doing the following check:

(Earliest time job 7 can start on stage 1 + processing time of job 7 on stage 1) + (Earliest time job 7 can start on stage 2 + processing time on stage 2)  $\geq$  Required delivery time – Tolerance

This comes out as:

$$(55+10) + (70+20) \geq 100 \text{ (or } 120 - 20)$$
$$90 \geq 100$$

Note that the comparison is only made with time the job 7 is expected to exit the system after stage 2. Hence as seen the time 90 is compared to the required delivery time minus the tolerance. The earliest start times for the first stage comes from the updated machine ready times from the previous iteration shown in table A19 of iteration 4, and the job ready times (which is zero in this case). The earliest starting times for job 7 on stage 2 is bigger of the time when job 7 finishes on stage 1 which is 65, and the time when job 7 can actually start after one of the machines becomes ready earliest on the second stage which is 70 (see table A17 of iteration 4 for machine ready times).

The result is false, meaning now is not the time for job 7 to be scheduled. So the job is not scheduled and the control now considers job 5 for a scheduling try. The same calculations are performed as described above for job 5, which will translate as:

$$(60+15) + (75 + 15) \geq 130 \text{ (or } 150 - 20) \text{ or}$$

90 >= 130

Note here that for earliest time job 5 can start on stage 1 is 60 because the tool for job 5 was available at 60, so it cannot start earlier than 60. On the second stage job 5 can start at 75 at the earliest because it finished at that time on stage 1. Result of the check is false, meaning job 5 cannot be scheduled now. At this point, the system selects jobs for scheduling according to the earliest delivery times. This makes job 7 schedulable first. This may result in jobs getting scheduled earlier, but still not later than their required times. Job 7 is scheduled on the required machines on stage 1 and stage 2 – namely machine 2 stage 1 and machine 1 on stage 2, and the machine ready times are updated as seen in Table A26. Note that job 7 will exit at time 90 (or 1:30:00.0000), but machine 1 on stage 2 will be ready at 120 (or 2:00:00.0000) because there is a maintenance scheduled on machine 1 from time 1:30:00.0000 to 2:00:00.0000.

**Table A22 Jobs at this iteration**

| Jobs | Stage 1 | Stage 2 | Path | Delivery time | Material\ Tool availability |
|------|---------|---------|------|---------------|-----------------------------|
| 5    | 15      | 15      |      | 2:30:00.000   | 1:00:00.000                 |
| 7    | 10      | 20      | 2,1  | 2:00:00.000   |                             |

**Table A23 Job ready times at stage - 1 ( $a_{i,1}$ )**

|           | Jobs         |   |
|-----------|--------------|---|
|           | 5            | 7 |
| $a_{i,1}$ | 1:00:00.0000 | 0 |

**Table A24 Earliest start times at stage 1 for all jobs**

| Jobs | $s_{i,1} \rightarrow$ Stage 1 |
|------|-------------------------------|
| 5    | 1:00:00.000                   |
| 7    | 55                            |

**Table A25 Earliest start times at stage 2 for all jobs**

| Jobs | $s_{i,2} \rightarrow$ Stage 2 |
|------|-------------------------------|
| 5    | 75                            |
| 7    | 70                            |

**Table A26 Updating machine ready times on stage 1 and stage 2 after iteration 5**

| Stage 1   |           | Stage 2   |           |
|-----------|-----------|-----------|-----------|
| Machine 1 | Machine 2 | Machine 1 | Machine 2 |
| 50        | 65        | 90 (+30)  | 75        |

**Table A27 Jobs scheduled list**

|       |
|-------|
| Job 3 |
| Job 4 |
| Job 1 |
| Job 2 |
| Job 7 |

### Iteration 6

In this iteration, job 5 is selected for scheduling on machines depending on when it becomes available and the earliest available machines, on machine 1 on stage 1 and machine 2 on stage 2. The machine ready times after scheduling job 5 are shown in Table A32.

**Table A28 Jobs in this iteration**

| Jobs | Stage 1 | Stage 2 | Path | Delivery time | Material\ Tool availability |
|------|---------|---------|------|---------------|-----------------------------|
| 5    | 15      | 15      |      | 2:30:00.000   | 1:00:00.000                 |

**Table A29 Job ready times at stage - 1 ( $a_{i,1}$ )**

|           | Jobs         |
|-----------|--------------|
|           | 5            |
| $a_{i,1}$ | 1:00:00.0000 |

**Table A30 Earliest start times at stage 1 for all jobs**

| Jobs | $s_{i,1} \rightarrow$ Stage 1 |
|------|-------------------------------|
| 5    | 1:00:00.000                   |

**Table A31 Earliest start times at stage 2 for all jobs**

| Jobs | $s_{i,2} \rightarrow$ Stage 2 |
|------|-------------------------------|
| 5    | 75                            |

**Table A32 Updating machine ready times on stage 1 and stage 2 after iteration 6**

| Stage 1   |           | Stage 2   |           |
|-----------|-----------|-----------|-----------|
| Machine 1 | Machine 2 | Machine 1 | Machine 2 |
| 75        | 65        | 120       | 90        |

**Table A33 Jobs scheduled list after this iteration**

|       |
|-------|
| Job 3 |
| Job 4 |
| Job 1 |
| Job 2 |
| Job 7 |
| Job 5 |

**Iteration 7**

At the beginning of this iteration, all standard jobs and special jobs 7 and 5 with delivery times have been scheduled. The remaining special jobs 6, 8, 9 and 10 are left. In this iteration, job 6 is selected for scheduling on machines on its required path. Note that machine 1 on stage 2 is maintained from time 90 until time 120. The updated machine ready times are shown in Table A37.

**Table A34 Job ready times at stage - 1 ( $a_{i,1}$ )**

|           | Jobs       |   |   |    |
|-----------|------------|---|---|----|
|           | 6          | 8 | 9 | 10 |
| $a_{i,1}$ | 30:00.0000 | 0 | 0 | 0  |

**Table A35 Earliest start times at stage 1 for all jobs**

| Jobs | $s_{i,1} \rightarrow$ Stage 1 |
|------|-------------------------------|
| 6    | 75                            |
| 8    | 75                            |
| 9    | 65                            |
| 10   | 75                            |

**Table A36 Earliest start times at stage 2 for all jobs**

| Jobs | $s_{i,2} \rightarrow$ Stage 2 |
|------|-------------------------------|
| 6    | 120                           |
| 8    | 90                            |
| 9    | 120                           |

|    |     |
|----|-----|
| 10 | 120 |
|----|-----|

**Table A37 Updating machine ready times on stage 1 and stage 2 after iteration 7**

| Stage 1   |           | Stage 2   |           |
|-----------|-----------|-----------|-----------|
| Machine 1 | Machine 2 | Machine 1 | Machine 2 |
| 105       | 65        | 140       | 90        |

**Table A38 Jobs scheduled list after this iteration**

|       |
|-------|
| Job 3 |
| Job 4 |
| Job 1 |
| Job 2 |
| Job 7 |
| Job 5 |
| Job 6 |

**Iteration 8**

In this iteration, job 8 is selected and scheduled on the required machine routing. The updated machine ready times are shown in Table A43.

**Table A39 Jobs for this iteration**

| Jobs | Stage 1 | Stage 2 | Path | Delivery time | Material\ Tool availability |
|------|---------|---------|------|---------------|-----------------------------|
| 8    | 15      | 30      | 1,2  |               |                             |
| 9    | 30      | 15      | 2,1  |               |                             |
| 10   | 15      | 15      | 1,1  |               |                             |

**Table A40 Job ready times at stage - 1 ( $a_{i,1}$ )**

|           | Jobs |   |    |
|-----------|------|---|----|
|           | 8    | 9 | 10 |
| $a_{i,1}$ | 0    | 0 | 0  |

**Table A41 Earliest start times at stage 1 for all jobs**

| Jobs | $s_{i,1} \rightarrow$ Stage 1 |
|------|-------------------------------|
| 8    | 105                           |
| 9    | 65                            |



|    |     |
|----|-----|
| 10 | 105 |
|----|-----|

**Table A42 Earliest start times at stage 2 for all jobs**

| Jobs | $s_{i,2} \rightarrow$ Stage 2 |
|------|-------------------------------|
| 8    | 120                           |
| 9    | 140                           |
| 10   | 140                           |

**Table A43 Updating machine ready times on stage 1 and stage 2 after iteration 8**

| Stage 1   |           | Stage 2   |           |
|-----------|-----------|-----------|-----------|
| Machine 1 | Machine 2 | Machine 1 | Machine 2 |
| 120       | 65        | 140       | 150       |

**Table A44 Jobs scheduled list after this iteration**

|       |
|-------|
| Job 3 |
| Job 4 |
| Job 1 |
| Job 2 |
| Job 7 |
| Job 5 |
| Job 6 |
| Job 8 |

**Iteration 9**

In this iteration, job 9 is selected for scheduling on the required machines. The machine ready times are shown in table A49.

**Table A45 Jobs in this iteration**

| Jobs | Stage 1 | Stage 2 | Path | Delivery time | Material\ Tool availability |
|------|---------|---------|------|---------------|-----------------------------|
| 9    | 30      | 15      | 2,1  |               |                             |
| 10   | 15      | 15      | 1,1  |               |                             |

**Table A46 Job ready times at stage - 1 ( $a_{i,1}$ )**

|  |             |
|--|-------------|
|  | <b>Jobs</b> |
|--|-------------|

|           |          |           |
|-----------|----------|-----------|
|           | <b>9</b> | <b>10</b> |
| $a_{i,1}$ | 0        | 0         |

**Table A47 Earliest start times at stage 1 for all jobs**

| <b>Jobs</b> | <b><math>s_{i,1} \rightarrow</math> Stage 1</b> |
|-------------|---|
| 9           | 65  |
| 10          | 120   |

**Table A48 Earliest start times at stage 2 for all jobs**

| <b>Jobs</b> | <b><math>s_{i,2} \rightarrow</math> Stage 2</b> |
|-------------|---|
| 9           | 140   |
| 10          | 140   |

**Table A49 Updating machine ready times on stage 1 and stage 2 after iteration 9**

| <b>Stage 1</b>   |                  | <b>Stage 2</b>   |                  |
|------------------|------------------|------------------|------------------|
| <b>Machine 1</b> | <b>Machine 2</b> | <b>Machine 1</b> | <b>Machine 2</b> |
| 120              | 95               | 155              | 150              |

**Table A50 Jobs scheduled list after this iteration**

|       |
|-------|
| Job 3 |
| Job 4 |
| Job 1 |
| Job 2 |
| Job 7 |
| Job 5 |
| Job 6 |
| Job 8 |
| Job 9 |

After scheduling job 10 in the next iteration, the machine ready times are updated in Table A51. After all the iterations, the time when each job will exit the system is shown in Table A52. As seen the special jobs with delivery times are delivered in time. A screen shot of the bounds and selection of jobs in the software is shown in Figure A1. As seen the bounds and the job selections match with the hand calculations.

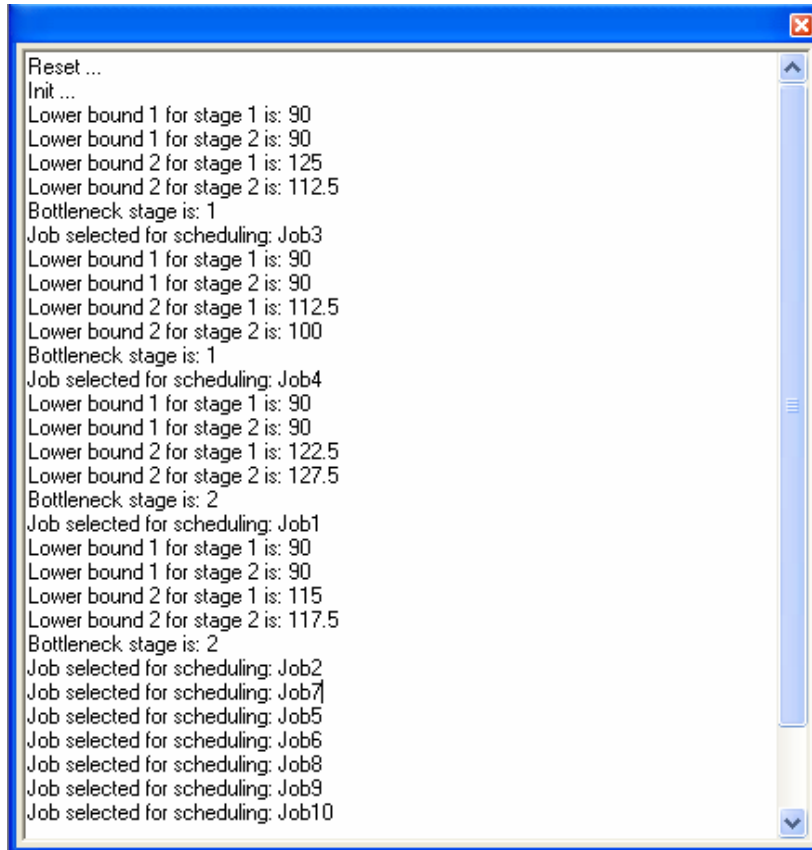
**Table A51 Updating machine ready times on stage 1 and stage 2 after iteration 10**

| <b>Stage 1</b>   |                  | <b>Stage 2</b>   |                  |
|------------------|------------------|------------------|------------------|
| <b>Machine 1</b> | <b>Machine 2</b> | <b>Machine 1</b> | <b>Machine 2</b> |
|                  |                  |                  |                  |

|     |     |     |     |
|-----|-----|-----|-----|
| 135 | 105 | 170 | 150 |
|-----|-----|-----|-----|

**Table A52 Job exit times for all jobs**

| <b>Jobs</b>       | 1  | 2  | 3  | 4  | 5  | 6   | 7   | 8   | 9   | 10  |
|-------------------|----|----|----|----|----|-----|-----|-----|-----|-----|
| <b>Exit times</b> | 70 | 75 | 50 | 50 | 90 | 140 | 120 | 150 | 155 | 170 |



**Figure A1 Results of bounds and selections in console of eM-Plant**

## APPENDIX 1.1: Detail events and results for simulation based FAM and validation

The reader is advised to refer to terminology such “Line 1” and “Line 2” described in the predictive simulation based FAM system concepts in section 5.2.2.1.1, to understand the following results. The system is solved by hand and validated by the simulation run. Because the calculations are based on instantaneous data during the simulation run, a simulation run is conducted, and the data recorded and compared to the hand calculations using the data at that moment. Figure A2, A3 and A4 show the screenshots of the results obtained by the simulation run. The detailed descriptions of the events and the calculations are shown as follows:

**Table A53 FAM analysis of Job 3**

|                             | Summation of processing times of jobs in buffer (a) | Processing time left for current job on machine (b) | Processing time for job 3 on stage 1 (c) | Total sum (a+b+c) | Jobs in buffer on this line |
|-----------------------------|---|---|--|-------------------|-----------------------------|
| <b>Line 1</b>               | 0   | 0   | 25                                       | 25                | 0                           |
| <b>Line 2</b>               | 0   | 0   | 25                                       | 25                | 0                           |
| <b>FAM condition: FALSE</b> |   |   |  |                   |                             |

In the FAM analysis, the optimality rule generator was used. In the analysis, the same sequence of jobs is used as generated by the optimization algorithm described in Appendix 1. The same system configuration and processing times data are used as shown in Table 5.1 to show the detailed results here. In the beginning, job 3 enters decision point 1 on stage 1. Refere table A53. Since this is the first job, there are no jobs on any machine on stage 1, and in the buffers, and hence the processing times are zero. The total sum of time on line 1 and line 2 are 25 minutes as seen in Table A53. Using the FAM rule logic, no rule is fired by the FAM, and the system keeps job 3 at its calculated position of going further to line 1.

**Table A54 FAM analysis of Job 4**

|                              | Summation of processing times of jobs in buffer (a) | Processing time left for current job on machine (b) | Processing time for job 4 on stage 1 (c) | Total sum (a+b+c) | Jobs in buffer on this line |
|------------------------------|---|---|--|-------------------|-----------------------------|
| <b>Line 1</b>                | 25  | 0   | 30                                       | 55                | Job 3                       |
| <b>Line 2</b>                | 0   | 0   | 30                                       | 30                | 0                           |
| <b>FAM condition = FALSE</b> |   |   |  |                   |                             |

After job 3, job 4 enters stage 1 and at the decision point is selected for FAM analysis. Refer Table A54. At this point it finds job 3 in the buffer on line 1. After doing calculations for optimality (described in section 5.2.2.1.1), using the total sum in Table A54, it is found out that job 4 should go to line 2, but this was also the result calculated by the optimization algorithm. Hence no overriding takes place for job 4.

**Table A55 FAM analysis of Job 1**

|                             | Summation of processing times of jobs in buffer (a) | Processing time left for current job on machine (b) | Processing time for job 1 on stage 1 (c) | Total sum (a+b+c) | Jobs in buffer on this line |
|-----------------------------|---|---|--|-------------------|-----------------------------|
| <b>Line 1</b>               | 0   | 25  | 25                                       | 50                | 0                           |
| <b>Line 2</b>               | 30  | 0   | 25                                       | 55                | Job 4                       |
| <b>FAM condition: FALSE</b> |   |   |  |                   |                             |

After that job 1 entered stage 1, and selected for FAM analysis. Refer to Table A55. The total sum is used for FAM analysis according to the FAM concept, and it is found that line 1 is the best result for job 1. Hence, no rule was fired and job 1 proceeds to line 1.

**Table A56 FAM analysis of Job 2**

|                             | Summation of processing times of jobs in buffer (a) | Processing time left for current job on machine (b) | Processing time for job 2 on stage 1 (c) | Total sum (a+b+c) | Jobs in buffer on this line |
|-----------------------------|---|---|--|-------------------|-----------------------------|
| <b>Line 1</b>               | 25  | 25  | 25                                       | 75                | Job 1                       |
| <b>Line 2</b>               | 0   | 30  | 25                                       | 55                | 0                           |
| <b>FAM condition: FALSE</b> |   |   |  |                   |                             |

Next, job 2 arrives at the decision point and is selected for FAM analysis. Refer to Table A56. Here job 1 is in buffer on line 1 and job 4 is being processed on the machine on line 2. Here too the FAM logic is used to determine if the condition is fulfilled using the total sum value. It is found out that it is not required to override the results of the optimization algorithm. Then, job 7 arrives in the system, but since job 7 is special it is not selected for analysis. So job 7 proceeds to its pre-calculated path.

**Table A57 FAM analysis of Job 3**

|                             | Summation of processing times of jobs in buffer (a) | Processing time left for current job on machine (b) | Processing time for job 3 on stage 2 (c) | Total sum (a+b+c) | Jobs in buffer on this line |
|-----------------------------|---|---|--|-------------------|-----------------------------|
| Line 1                      | 0   | 0   | 25                                       | 25                | 0                           |
| Line 2                      | 0   | 0   | 25                                       | 25                | 0                           |
| <b>FAM condition: FALSE</b> |   |   |  |                   |                             |

Next job 3 finishes processing on stage 1 and prepares to go to stage 2, where it is again selected for FAM analysis. Refer Table A57 for the details. Here since there are no other jobs on stage 2, in the buffer and on the machines, the values of processing times are accordingly zero. Upon conducting the condition check, it is found that job 3 can stick to its original pre-calculated path.

**Table A58 FAM analysis of Job 4**

|                             | Summation of processing times of jobs in buffer (a) | Processing time left for current job on machine (b) | Processing time for job 4 on stage 2 (c) | Total sum (a+b+c) | Jobs in buffer on this line |
|-----------------------------|---|---|--|-------------------|-----------------------------|
| Line 1                      | 0   | 20  | 20                                       | 40                | 0                           |
| Line 2                      | 0   | 0   | 0  | 20                | 0                           |
| <b>FAM condition: FALSE</b> |   |   |  |                   |                             |

Then, job 4 finished on stage 1 and begins to enter stage 2, where it is selected for FAM analysis. Refer Table A58, with all the instantaneous data filled in. Upon condition check, no rule is fired by the FAM and so job 4 proceeds to its original path.

**Table A59 FAM analysis of Job 1**

|                             | Summation of processing times of jobs in buffer (a) | Processing time left for current job on machine (b) | Processing time for job 1 on stage 2 (c) | Total sum (a+b+c) | Jobs in buffer on this line |
|-----------------------------|---|---|--|-------------------|-----------------------------|
| Line 1                      | 0   | 0   | 20                                       | 20                | 0                           |
| Line 2                      | 0   | 0   | 20                                       | 20                | 0                           |
| <b>FAM condition: FALSE</b> |   |   |  |                   |                             |

Next job 1 tries to enter stage 2 and is selected for FAM analysis. Refer Table A59 for the detailed instantaneous data. There were no jobs on the second stage when job 1 came in for analysis, because job 4 and job 3 were already finished. Upon analysis it is found that job 1 can take its pre-calculated path from the optimization algorithm, or in other words the FAM condition was false.

**Table A60 FAM analysis of Job 2**

|                             | <b>Summation of processing times of jobs in buffer (a)</b> | <b>Processing time left for current job on machine (b)</b> | <b>Processing time for job 2 on stage 2 (c)</b> | <b>Total sum (a+b+c)</b> | <b>Jobs in buffer on this line</b> |
|-----------------------------|--|--|---|--------------------------|------------------------------------|
| <b>Line 1</b>               | 15   | 0  | 20  | 35                       | Job 1                              |
| <b>Line 2</b>               | 0  | 0  | 20  | 20                       | 0                                  |
| <b>FAM condition: FALSE</b> |  |  |   |                          |                                    |

Then job 2 was finished on stage 1 and it prepares to go to stage 2. Refer Table A60 for the detailed instantaneous data used in the FAM analysis. Upon analysis, it is found that job 2 can take its pre-calculated path.

**Table A61 FAM analysis of Job 5**

|                             | <b>Summation of processing times of jobs in buffer (a)</b> | <b>Processing time left for current job on machine (b)</b> | <b>Processing time for job 5 on stage 1 (c)</b> | <b>Total sum (a+b+c)</b> | <b>Jobs in buffer on this line</b> |
|-----------------------------|--|--|---|--------------------------|------------------------------------|
| <b>Line 1</b>               | 0  | 0  | 15  | 15                       | 0                                  |
| <b>Line 2</b>               | 0  | 5  | 15  | 20                       | 0                                  |
| <b>FAM condition: FALSE</b> |  |  |   |                          |                                    |

Then, job 5 enters stage 1 during the run-time clock of the simulation. At that time job 7 is being processed on machine 2 on stage 1. Refer Table A61. Note that job 5 is a special job, with only delivery times !! Hence, this makes it possible to be considered “standard” for FAM analysis, since the FAM only tries to reduce the makespan of all the considered jobs – thus not affecting the delivery time results negatively. There were no jobs in the buffer on stage 1 at that instant. Refer Table A61 for analysis data. Upon analysis, it is found out that overriding is not required because the FAM condition was false.

Then the next event was the incoming of job 6 in the system at the entrance. It enters the decision point and is selected for analysis. Since job 6 is special, it is not

analysed, and it goes on its pre-calculated path in the system.

Then the next event was in the incoming of job 8 in the system at the entrance. It enters the decision point and is selected for analysis. Since job 8 is special, it is not analysed, and it goes on its pre-calculated path in the system. Then job 7 enters stage 2 after finishing on stage 1, and since it is a special job it is not selected for FAM analysis at the decision point for the second stage.

**Table A62 FAM analysis of Job 5**

|                             | <b>Summation of processing times of jobs in buffer (a)</b> | <b>Processing time left for current job on machine (b)</b> | <b>Processing time for job 5 on stage 2 (c)</b> | <b>Total sum (a+b+c)</b> | <b>Jobs in buffer on this line</b> |
|-----------------------------|--|--|---|--------------------------|------------------------------------|
| <b>Line 1</b>               | 0  | 15   | 15  | 60                       | 0                                  |
| <b>Line 2</b>               | 0  | 0  | 15  | 15                       | 0                                  |
| <b>FAM condition: FALSE</b> |  |  |   |                          |                                    |

Then job 5 enters stage 2 and is selected for FAM analysis. Refer Table A62. As mentioned earlier, job 5 is special according to the definition, but since it does not have routing constraints, it is considered “standard” during the FAM analysis. Note during this time, machine 1 on line 1 was maintained for a duration of 30 minutes which is added to the total sum during the FAM analysis. Again, upon analysis, it is found out that the FAM condition is false, which makes the rule unfirable. Job 5 hence follows its pre-calculated path.

Then job 6 arrives at the second stage and is selected for FAM analysis. Since job 6 is special, nothing is done with it, and it proceeds to its pre-calculated path.

Then job 8 arrives at the second stage and is selected for FAM analysis. Since job 8 is special, nothing is done with it, and it proceeds to its pre-calculated path.

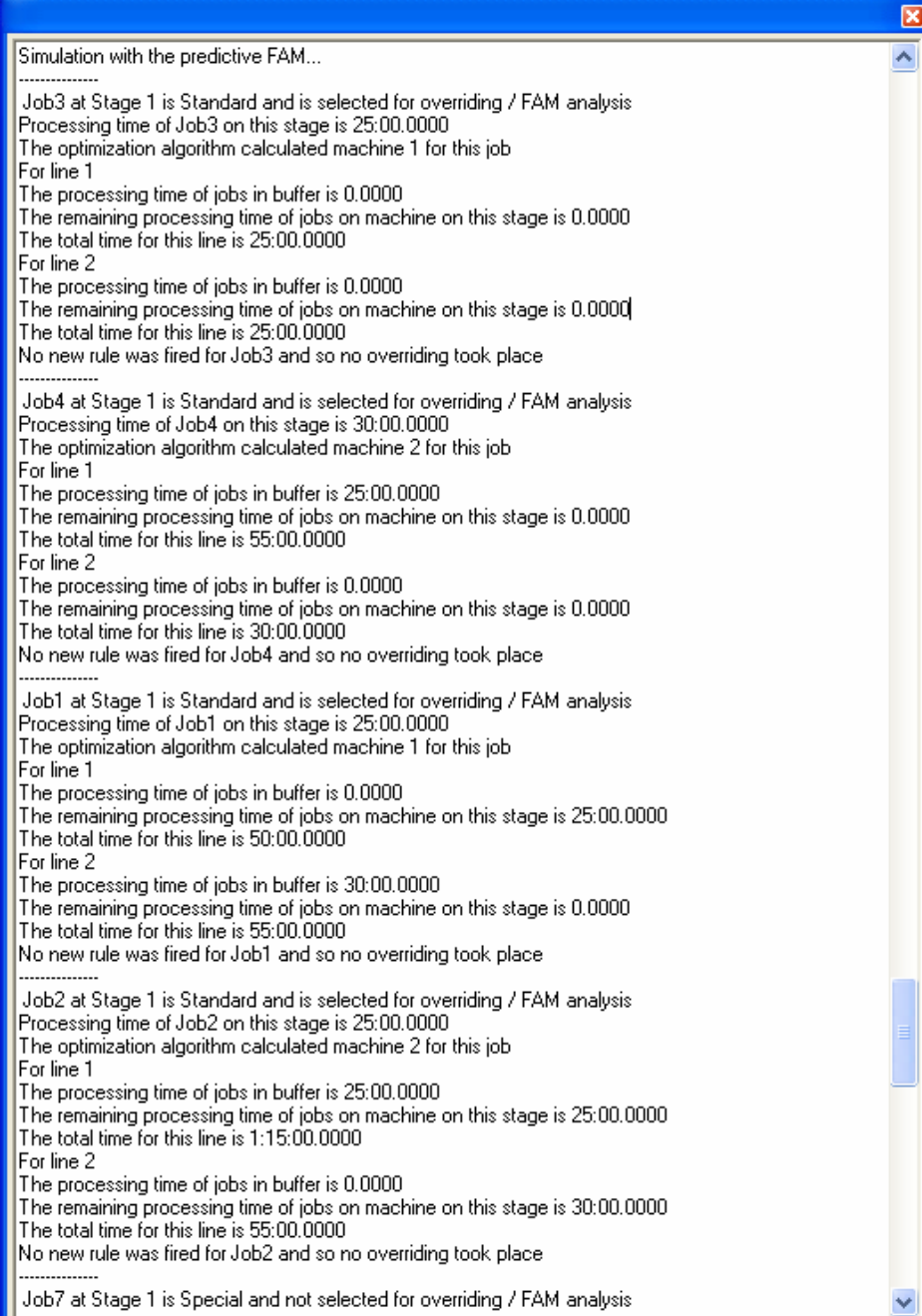
Then job 9 arrives at the first stage and is selected for FAM analysis. Since job 9 is special, nothing is done with it, and it proceeds to its pre-calculated path.

Then job 10 arrives at the first stage and is selected for FAM analysis. Since job 10 is special, nothing is done with it, and it proceeds to its pre-calculated path.

Then job 10 arrives at the second stage and is selected for FAM analysis. Since job 10 is special, nothing is done with it, and it proceeds to its pre-calculated path.



Then job 9 arrives at the second stage and is selected for FAM analysis. Since job 9 is special, nothing is done with it, and it proceeds to its pre-calculated path.



```
Simulation with the predictive FAM...
-----
Job3 at Stage 1 is Standard and is selected for overriding / FAM analysis
Processing time of Job3 on this stage is 25:00.0000
The optimization algorithm calculated machine 1 for this job
For line 1
The processing time of jobs in buffer is 0.0000
The remaining processing time of jobs on machine on this stage is 0.0000
The total time for this line is 25:00.0000
For line 2
The processing time of jobs in buffer is 0.0000
The remaining processing time of jobs on machine on this stage is 0.0000
The total time for this line is 25:00.0000
No new rule was fired for Job3 and so no overriding took place
-----
Job4 at Stage 1 is Standard and is selected for overriding / FAM analysis
Processing time of Job4 on this stage is 30:00.0000
The optimization algorithm calculated machine 2 for this job
For line 1
The processing time of jobs in buffer is 25:00.0000
The remaining processing time of jobs on machine on this stage is 0.0000
The total time for this line is 55:00.0000
For line 2
The processing time of jobs in buffer is 0.0000
The remaining processing time of jobs on machine on this stage is 0.0000
The total time for this line is 30:00.0000
No new rule was fired for Job4 and so no overriding took place
-----
Job1 at Stage 1 is Standard and is selected for overriding / FAM analysis
Processing time of Job1 on this stage is 25:00.0000
The optimization algorithm calculated machine 1 for this job
For line 1
The processing time of jobs in buffer is 0.0000
The remaining processing time of jobs on machine on this stage is 25:00.0000
The total time for this line is 50:00.0000
For line 2
The processing time of jobs in buffer is 30:00.0000
The remaining processing time of jobs on machine on this stage is 0.0000
The total time for this line is 55:00.0000
No new rule was fired for Job1 and so no overriding took place
-----
Job2 at Stage 1 is Standard and is selected for overriding / FAM analysis
Processing time of Job2 on this stage is 25:00.0000
The optimization algorithm calculated machine 2 for this job
For line 1
The processing time of jobs in buffer is 25:00.0000
The remaining processing time of jobs on machine on this stage is 25:00.0000
The total time for this line is 1:15:00.0000
For line 2
The processing time of jobs in buffer is 0.0000
The remaining processing time of jobs on machine on this stage is 30:00.0000
The total time for this line is 55:00.0000
No new rule was fired for Job2 and so no overriding took place
-----
Job7 at Stage 1 is Special and not selected for overriding / FAM analysis
```

**Figure A2 Run-time results of the FAM in eM-Plant console: Part 1**

```
-----  
Job7 at Stage 1 is Special and not selected for overriding / FAM analysis  
-----  
Job3 at Stage 2 is Standard and is selected for overriding / FAM analysis  
Processing time of Job3 on this stage is 25:00.0000  
The optimization algorithm calculated machine 1 for this job  
For line 1  
The processing time of jobs in buffer is 0.0000  
The remaining processing time of jobs on machine on this stage is 0.0000  
The total time for this line is 25:00.0000  
For line 2  
The processing time of jobs in buffer is 0.0000  
The remaining processing time of jobs on machine on this stage is 0.0000  
The total time for this line is 25:00.0000  
No new rule was fired for Job3 and so no overriding took place  
-----  
Job4 at Stage 2 is Standard and is selected for overriding / FAM analysis  
Processing time of Job4 on this stage is 20:00.0000  
The optimization algorithm calculated machine 2 for this job  
For line 1  
The processing time of jobs in buffer is 0.0000  
The remaining processing time of jobs on machine on this stage is 20:00.0000  
The total time for this line is 40:00.0000  
For line 2  
The processing time of jobs in buffer is 0.0000  
The remaining processing time of jobs on machine on this stage is 0.0000  
The total time for this line is 20:00.0000  
No new rule was fired for Job4 and so no overriding took place  
-----  
Job1 at Stage 2 is Standard and is selected for overriding / FAM analysis  
Processing time of Job1 on this stage is 20:00.0000  
The optimization algorithm calculated machine 1 for this job  
For line 1  
The processing time of jobs in buffer is 0.0000  
The remaining processing time of jobs on machine on this stage is 0.0000  
The total time for this line is 20:00.0000  
For line 2  
The processing time of jobs in buffer is 0.0000  
The remaining processing time of jobs on machine on this stage is 0.0000  
The total time for this line is 20:00.0000  
No new rule was fired for Job1 and so no overriding took place  
-----  
Job2 at Stage 2 is Standard and is selected for overriding / FAM analysis  
Processing time of Job2 on this stage is 20:00.0000  
The optimization algorithm calculated machine 2 for this job  
For line 1  
The processing time of jobs in buffer is 0.0000  
The remaining processing time of jobs on machine on this stage is 15:00.0000  
The total time for this line is 35:00.0000  
For line 2  
The processing time of jobs in buffer is 0.0000  
The remaining processing time of jobs on machine on this stage is 0.0000  
The total time for this line is 20:00.0000  
No new rule was fired for Job2 and so no overriding took place  
-----
```

**Figure A3 Run-time results of the simulation based FAM in eM-Plant console:  
Part 2**

```
No new rule was fired for Job2 and so no overriding took place
-----
Job5 at Stage 1 is Standard and is selected for overriding / FAM analysis
Processing time of Job5 on this stage is 15:00.0000
The optimization algorithm calculated machine 1 for this job
For line 1
The processing time of jobs in buffer is 0.0000
The remaining processing time of jobs on machine on this stage is 0.0000
The total time for this line is 15:00.0000
For line 2
The processing time of jobs in buffer is 0.0000
The remaining processing time of jobs on machine on this stage is 5:00.0000
The total time for this line is 20:00.0000
No new rule was fired for Job5 and so no overriding took place
-----
Job6 at Stage 1 is Special and not selected for overriding / FAM analysis
-----
Job8 at Stage 1 is Special and not selected for overriding / FAM analysis
-----
Job7 at Stage 2 is Special and not selected for overriding / FAM analysis
-----
Job5 at Stage 2 is Standard and is selected for overriding / FAM analysis
Processing time of Job5 on this stage is 15:00.0000
The optimization algorithm calculated machine 2 for this job
For line 1
The processing time of jobs in buffer is 0.0000
The remaining processing time of jobs on machine on this stage is 15:00.0000
Machine was maintained for a duration 30:00.0000
The total time for this line is 1:00:00.0000
For line 2
The processing time of jobs in buffer is 0.0000
The remaining processing time of jobs on machine on this stage is 0.0000
The total time for this line is 15:00.0000
No new rule was fired for Job5 and so no overriding took place
-----
Job6 at Stage 2 is Special and not selected for overriding / FAM analysis
-----
Job8 at Stage 2 is Special and not selected for overriding / FAM analysis
-----
Job9 at Stage 1 is Special and not selected for overriding / FAM analysis
-----
Job10 at Stage 1 is Special and not selected for overriding / FAM analysis
-----
Job10 at Stage 2 is Special and not selected for overriding / FAM analysis
-----
Job9 at Stage 2 is Special and not selected for overriding / FAM analysis
Reset ...
```

**Figure A4 Run-time results of the simulation based FAM in eM-Plant console:  
Part 3**

## APPENDIX 2: Detail iterations for the match-up rescheduling algorithm and validation

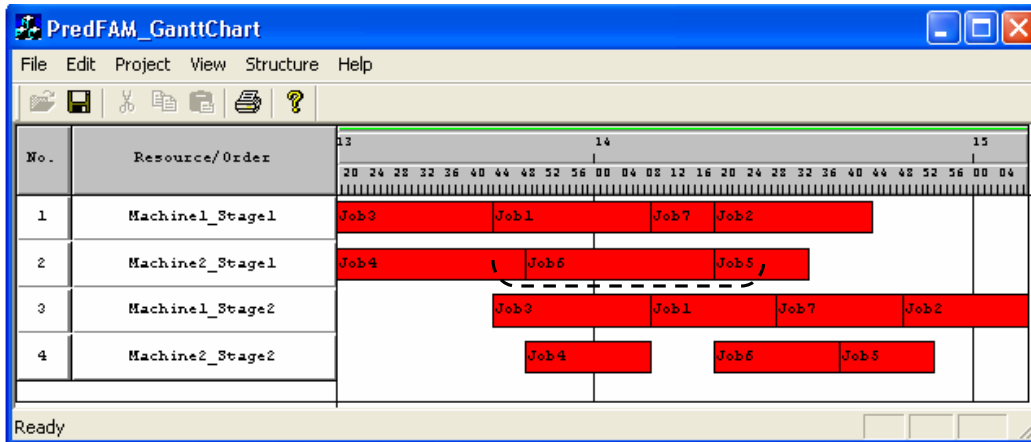


Figure A5 Predictive FAM schedule gantt chart

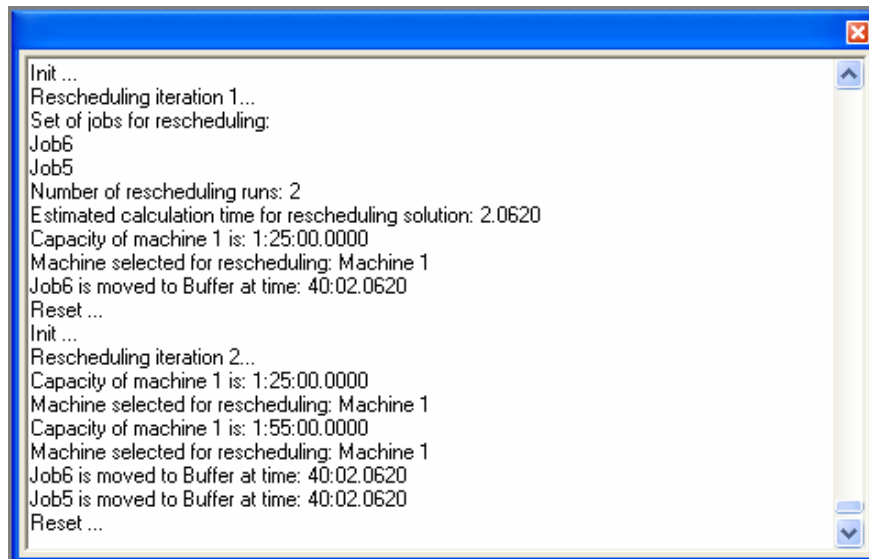


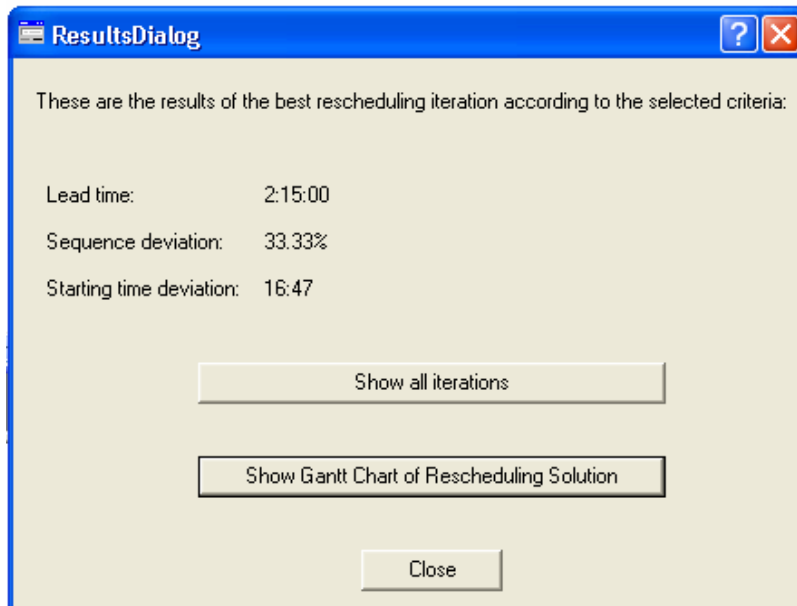
Figure A6 Detailed iterations and selections in eM-Plant console

Table A63 Detailed results on makespan

| Method                | Makespan |
|-----------------------|----------|
| Predictive schedule   | 110      |
| Upper bounds          | 135      |
| Rescheduling solution | 135      |

**Table A64 Results on performance indicators**

| <b>Method</b>                   | <b>Starting time deviations</b> | <b>Sequence deviations</b> |
|---------------------------------|---------------------------------|----------------------------|
| Selected rescheduling iteration | 16                              | 33 %                       |



**Figure A7 Results from the software run**

### APPENDIX 3: Detail iterations for the selective re-routing algorithm and validation

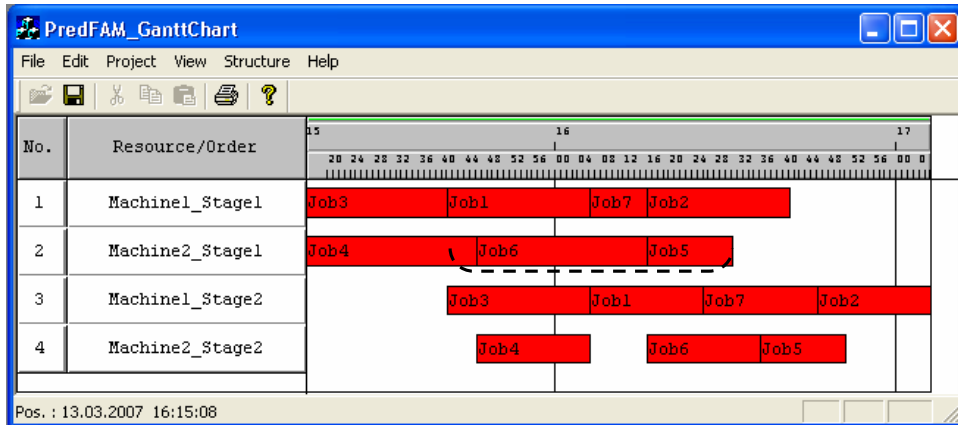


Figure A8 Predictive FAM gantt chart

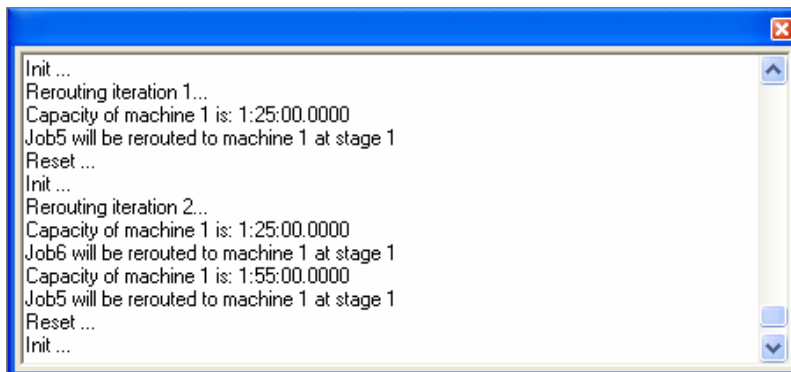


Figure A9 Detailed iterations and selections in eM-Plant console

Table A65 Detailed results on makespan

| Method                | Makespan |
|-----------------------|----------|
| Predictive schedule   | 110      |
| Upper bounds          | 150      |
| Rescheduling solution | 145      |

Table A66 Results on performance indicators

| Method                          | Starting time deviations | Sequence deviations |
|---------------------------------|--------------------------|---------------------|
| Selected rescheduling iteration | 23                       | 66 %                |

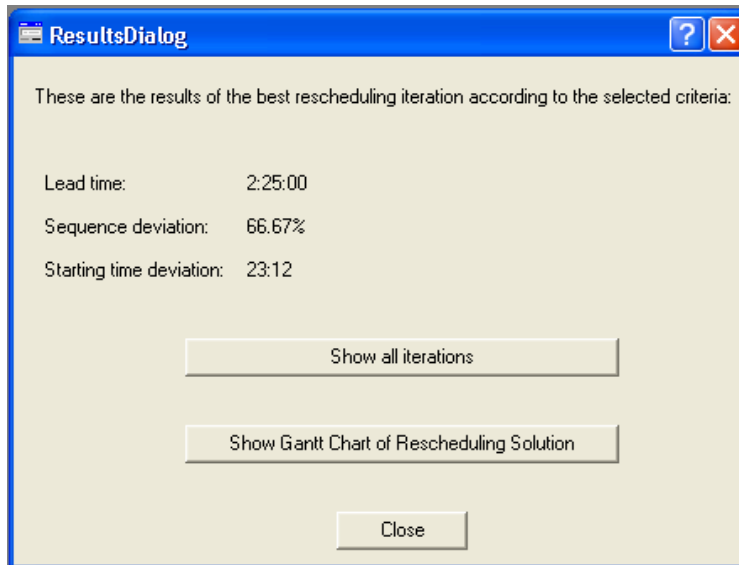


Figure A10 Results from the software run

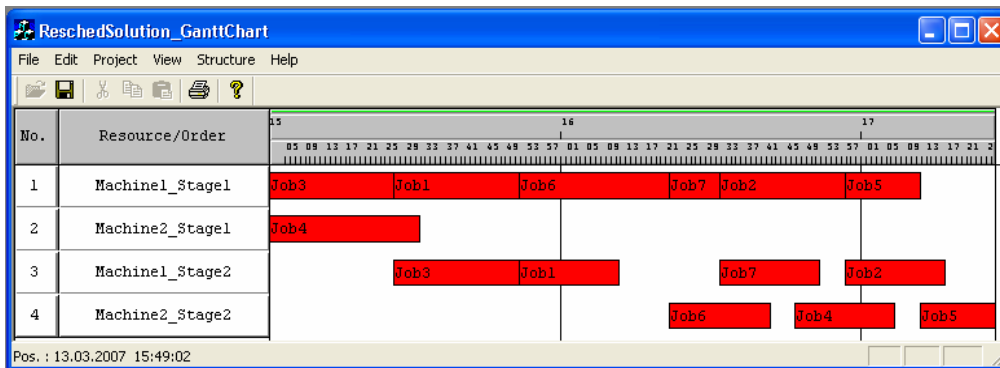


Figure A11 Rescheduling solution gantt chart

## APPENDIX 4: Methods, Tables and Variable objects used during implementation

The following objects were used during the implementation of the system. The classes and the changes that were made compared to their ancestor classes are described below:

### Tables

- **SpecialJobs (derived from SpecialJobs)**  
For every special job this table contains its path and its position in the entry sequence, i.e. the number of the row where it will be entered in the “DeliveryTable”. This table is edited by the user.
- **DeliveryTable (derived from DeliveryTable)**  
This table is used by the source of the system in order to determine the sequence in which the jobs enter the system. This table is filled by the “Scheduling\_Algorithm” method.
- **ToolAvailability (derived from ToolAvailability)**  
For every job, this table contains the time when the tool will be available. This table is edited by the user.
- **MaintenanceTimes (derived from MaintenanceTimes)**  
Using this table the user can set the starting time and the ending time of a scheduled maintenance on one or more of the machines in the system.
- **Material availability (derived from MaterialAvailability)**  
This table contains the time, when the materials for certain jobs will be available in the planning horizon.
- **DecisionPoints (derived from DecisionPoints)**  
This table contains the information which rule generators shall be used at which decision point. This table is edited by the user.
- **MachineCount (derived from MachineCount)**  
This table contains information about the number of machines at each stage. This table is filled by the “CountStagesAndMachines” method.



- **Times\_StageX (derived from Times\_Stage)**  
One of these tables should exist for each stage. They contain the processing times of the jobs for each stage. These tables can be filled by the “RandomTimes” method and edited by the user.
- **JobSetOutTimes\_StageX (derived from JobSetOutTime)**  
One of these tables should exist for each stage. They contain information about how much time would be needed for the different jobs to take them out of the buffers of this stage.
- **JobTransportationTimes\_StageX (derived from JobTransportationTime)**  
One of these tables should exist for each stage. For every job they contain information about the amount of time that would be needed to transport it from its original machine to every other machine at this stage.
- **JobSetInTimes\_StageX (derived from JobSetInTime)**  
One of these tables should exist for each stage. They contain information about how much time would be needed for the different jobs to insert them into the buffers of this stage.
- **JobExitTimes (derived from JobExitTimes)**  
This table saves the time points when the jobs left the system when the pure predictive, the predictive FAM or a random schedule is used. It also records the job exit times for the simulation run that simulates the predictive FAM schedule under the influence of an exception. This table is filled by the “JobExitRecorder” method.
- **ReschedulingJobExitTimes (derived from ReschedulingJobExitTimes)**  
This table saves the time points when the jobs left the system during the rescheduling iterations and the simulation run that analyses the selected iteration with the reactive FAM.
- **Rescheduling\_JobStartingTimes (derived from JobStartingTimes)**  
This table saves the starting times for every job on every machine during a rescheduling iteration. It is filled by the “StartTimeRecorder” method.
- **Upper\_bound\_JobStartingTimes (derived from JobStartingTimes)**  
This table saves the starting times for every job on every machine during the simulation of the predictive FAM under the influence of an exception. It is filled by the “StartTimeRecorder” method.

- **TempResults (derived from TempResults)**  
 This table saves the exiting time of every job for a single rescheduling iteration. After the iteration has finished the contents of the table are written to the “ReschedulingJobExitTimes” table. The reason for this approach is as follows: In order to determine the column of the “ReschedulingJobExitTimes” table where the exiting times should be saved, the number of rescheduling runs that will be performed is needed. This number will be calculated by the “MatchupReschedulingAlgorithm” method. But since it is possible that jobs exit the system, before the method is called, the exiting times of the jobs will first be written to the “TempResults” table.
- **JobsToReschedule (derived from JobsToReschedule)**  
 During the first rescheduling iteration the “MatchupReschedulingAlgorithm” method saves the jobs that are candidates for rescheduling in this table. During the rescheduling iterations the jobs are selected from this table and do not have to be computed again.
- **JobQueue (derived from JobQueue)**  
 If a rescheduled job can not enter its destination buffer right away, it is written to this table by the “ShiftJob” method. The “InsertJobsFromQueue” method picks jobs from this table and inserts them into their destination buffer, as soon as they have capacity for an additional job.
- **ReschedulingResults (derived from ReschedulingResults)**  
 This table saves the make span, sequence deviation and starting time deviation for each rescheduling iteration and the reactive FAM run.
- **ReschedulingMoves (derived from ReschedulingMoves)**  
 This table saves the rescheduling moves that were made in the rescheduling iterations. This information is used by the failure handler during the reactive FAM run, so that the “MatchupReschedulingAlgorithm” does not have to be called again.
- **Re-routingMoves (derived from ReroutingMoves)**  
 This table saves the re-routing moves that were made in the rescheduling iterations using the selective re-routing algorithm. This information is used by the failure handler during the reactive FAM run, so that the algorithm does not have to be called again.

- **ReactiveFAM\_JobStartingTimes (derived from JobStartingTimes)**  
This table saves the job starting times during the simulation run with the reactive FAM. It is filled by the “StartTimeRecorder” method.
- **ReschedSolution\_JobStartingTimes (derived from JobStartingTimes)**  
This table saves the job starting times during the simulation run of the selected rescheduling iteration. It is filled by the “StartTimeRecorder” method.
- **FAM\_JobStartingTimes (derived from JobStartingTimes)**  
This table saves the job starting times during the simulation run of the predictive FAM. It is filled by the “StartTimeRecorder” method.
- **HeuristicSchedule (derived from ScheduleTable)**  
This table saves the paths for all jobs that are calculated by the “Scheduling\_Algorithm” method.
- **PredFAMRoutings (derived from ScheduleTable)**  
This table contains the paths for all jobs after the predictive FAM run has been completed. These paths consist of the calculations of the “Scheduling\_Algorithm” method and the changes that the rule generators calculated during the predictive FAM run.
- **ReschedRoutings (derived from Schedule Table)**  
This table contains the routings for the simulation run with the selective re-routing algorithm.
- **RandomRoutings (derived from ScheduleTable)**  
This table saves paths for all jobs that were calculated by the “RandomScheduling” method.
- **Jobs\_StageX\_MachineX (derived from Jobs\_Stage\_Machine)**  
One of these tables should exist for each machine. For every machine they save the names of the jobs that are processed and time points when the jobs start on the machines. These tables are filled by the “StartTimeRecorder” method.

### Variables

- **NumberOfJobs**  
This variable contains the number of jobs the user wishes to simulate.

- **NumberOfStages**  
This variable contains the number of stages the user has modelled.
- **NumberOfPredictiveFAMReschedulings**  
This variable saves the number of changes to the pure predictive schedule the rule generators have made during the predictive FAM run.
- **NumberOfReactiveFAMReschedulings**  
This variable saves the number of changes to the predictive FAM schedule the bottleneck rule has made during the reactive FAM run.
- **ExceptionType**  
This variable saves the type of exception that happened.
- **ExceptionTime**  
This variable saves the time point when an exception happened.
- **ExceptionDuration**  
This variable saves the time span the exception will last.
- **ExceptionLocation**  
This variable saves the location of the exception occurred.
- **Upper\_bound\_makespan**  
This variable saves the makespan of the simulation of the predictive FAM schedule under the influence of the exception.
- **Upper\_bound\_starting\_time\_deviation**  
This variable saves the starting time deviation of the simulation of the predictive FAM schedule under the influence of the exception.
- **Upper\_bound\_sequence\_deviation**  
This variable saves the sequence deviation of the simulation of the predictive FAM schedule under the influence of the exception.
- **Exception**  
As soon as an exception occurs, this variable is set to true. This is needed to control the workflow of the whole system.

- **ReschedulingRun**  
This variable contains the number of the current rescheduling run. This is needed to control the workflow of the whole system.
- **SelectedReschedulingRun**  
After all rescheduling iterations are finished, this variable will contain the number of the iteration which is best according to the criterion the user has selected.
- **NumberOfReschedulingRuns**  
After the “MatchupReschedulingAlgorithm” method has been executed for the first time, this variable will contain the number of rescheduling iterations that will be carried out by the system.
- **State**  
This variable contains information about the current state of the whole system. This information is encoded as an integer and is used to control the workflow of the whole system.

## **Methods**

- **InitializeNewRun (Derived from Method)**  
Initializes the whole system for a new simulation. It calls the methods “CountStagesAndMachines”, “ClearTables”, “DeleteDecisionRules”, “InsertException”, “ResetVariables” and “init”.
- **MatchupReschedulingAlgorithm (Derived from Method)**  
This method implements the match up rescheduling algorithm and is called by the failure handler during each rescheduling iteration. It calculates the machines where to reschedule the jobs during the rescheduling iterations. It calls the “ShiftJob” method.
- **SelectiveReroutingAlgorithm (Derived from Method)**  
This method implements the Selective Rerouting Algorithm and is called by the failure handler during each rescheduling iteration. It calculates the machines where to reschedule the jobs during the rescheduling iterations. It calls the “ShiftJob” method.
- **SchedulingAlgorithm (Derived from Method)**

This method implements the predictive heuristic and calculates the pure predictive schedule. It call the methods “DeleteOldJobs” and “MakeNewJobs”.

- **RandomScheduling (Derived from Method)**  
This method calculates a random schedule.
- **StartTimeRecorder (Derived from Method)**  
This method is used as an entry control on the machines in order to record the job starting times.
- **InsertBottleneckRuleForFAMResched (Derived from Method)**  
This method inserts the bottleneck rule during the reactive FAM run, by setting the method „FAM\_Bottleneck\_Rule“ as an entry control for all decision points.
- **FAM\_Buffer\_Rule (Derived from Method)**  
This method implements the buffer rule. It is used as an entry control of the decision points during the predictive FAM run.
- **FAM\_Bottleneck\_Rule (Derived from Method)**  
This method implements the bottleneck rule. It is used as an entry control of the decision points during the predictive FAM run or the reactive FAM run.
- **FAM\_PathCopier (Derived from Method)**  
This method is needed during the predictive FAM run, if the user has not selected a rule generator for a decision point. Then this method is inserted as an entry control, in order to copy the pure predictive path of the job that triggered the entry control to the predictive FAM schedule.
- **EntranceSemaphore (Derived from Method)**  
This method is inserted as an entry control of a buffer, where is job is about to be rescheduled. During the insertion process together with the exit semaphore it ensures, that the buffer will have a capacity of at least 1.
- **ExitSemaphore (Derived from Method)**  
This method is inserted as an exit control of a buffer, where is job is about to be rescheduled. During the insertion process together with the entrance semaphore it ensures, that the buffer will have a capacity of at least 1.
- **ShiftJob (Derived from Method)**

This method executes the actual job shift in the system. It is called by the “MatchupReschedulingAlgorithm” method. After the set out time of the rescheduled job it deletes the job from the system and then waits for an amount of time that is equal to the time needed to transport the job from its original to its new machine. If it finds the destination buffer full after this time, it inserts the method “InsertJobsFromQueue” as an exit control into the destination buffer. This control will insert the job as soon as the buffer has capacity again. Otherwise the entrance and exit semaphores are inserted and the process of inserting the job in the buffer is started.

- **DeleteOldJobs (Derived from Method)**  
This method is called by the “Scheduling \_Algorithm” method. It deletes all job objects from the class library.
- **MakeNewJobs (Derived from Method)**  
This method is called by the “Scheduling \_Algorithm” method. It creates as many new job objects in the class library as the user has defined in the scheduling dialog.
- **InsertDecisionRules (Derived from Method)**  
This method is called by the “init” method before the predictive FAM run. It inserts the rule generators as entrance controls in the decision points according to the specifications the user has entered in the decision points table.
- **DeleteDecisionRules (Derived from Method)**  
This method deletes all rule generators from the decision points.
- **Router (Derived from Method)**  
This method is used by the flow controller objects. It looks up the path of the job that triggered the method in the correct schedule and returns the value.
- **RandomTimes (Derived from Method)**  
This method fills the tables that contain the processing times with random values. These are generated by a uniform distribution.
- **CountStagesAndMachines (Derived from Method)**  
This method counts the number of stages in the model and the number of machines on each stage.
- **InsertException (Derived from Method)**

This method inserts the exception starting time and the exception duration in the machine specified in the variable "ExceptionLocation".

- **DeleteException (Derived from Method)**  
This method deletes any exception from the machine specified by the variable „ExceptionLocation“.
- **ClearTables (Derived from Method)**  
This method initializes all tables for a new simulation.
- **FailureHandler (Derived from Method)**  
This method is called as soon as a exception happens. During the rescheduling iterations it triggers the "MatchupReschedulingAlgorithm" method. During the reactive FAM run it looks up the calculated job shifts of the selected rescheduling iteration and performs the job shifts by calling the "ShiftJob" method.
- **ShowResults (Derived from Method)**  
This method writes the results of the selected rescheduling run modified by the reactive FAM, if the user has selected this analysation, to the results dialog and displays it.
- **JobExitRecorder (Derived from Method)**  
This method is used as an entrance control of the sink. It records the times when the jobs exit the system.
- **InsertJobsFromQueue (Derived from Method)**  
This method is inserted as an exit control to a buffer where a job should be rescheduled and that was full as soon as the job arrived at the buffer. As soon as a job exits the buffer, this method inserts the entrance and exit semaphores and starts the insertion process of the job that could not be inserted before.
- **ResetVariables (Derived from Method)**  
This method resets all variables to their initial values before a new simulation run.
- **CalculateReschedulingResults (Derived from Method)**  
This method calculates and saves the make span, the starting time deviation and the sequence deviation of a rescheduling iteration.



- **CalculateUpperBound (Derived from Method)**  
This method calculates and saves the make span, starting time deviation and the sequence deviation of the exception run.
- **InsertMaintenance (Derived from Method)**  
This method schedules the start and stop maintenance methods, i.e. this method determines when the methods are called.
- **InitializeObjects (Derived from Method)**  
This method connects source to the delivery table, writes the failure handler to the failure control of the processing units, it connects the start time recorders as entry control for the single processing units. It also sets the job exit recorder as entry controller for the drain. It sets the router method as a selection method for the flow controllers.
- **MakeTables (Derived from Method)**  
This method adjusts the number of tables required for job set in times, the job set out time, transportation time, the number of the processing time tables, and the Jobs\_StageX\_MachineX tables.
- **StartMaintenance (Derived from Method)**  
This method sets the machine to pause when this method is called by the insert maintenance method.
- **StopMaintenance (Derived from Method)**  
This method sets the machine to working again when this method is called by the insert maintenance method.
- **Init (Derived from Method)**  
This method initializes the system for the next run of the whole simulation. For example it adjusts the state variable of the system.
- **Endsim (Derived from Method)**  
This method triggers all the calculations that need to be done after a simulation run. Also it triggers a new simulation run if the whole simulation is not finished yet.

#### Other generic objects

- **Source (Derived from Source)**

The “Time of Creation” was changed to “Delivery Table”. The name of the according table, namely “DeliveryTable”, was already entered in the “Table”-textbox.

- **Drain (Derived from Drain)**  
The processing time was changed to 0.
- **Buffer (Derived from Buffer)**  
The capacity was changed to 99 and the processing time set to 0.
- **FlowControl (Derived from FlowControl)**  
The strategy was set to “Attribute” and the attribute type was defined as “Integer”.
- **Dialog (Derived from Dialog)**  
This dialog was developed to give the user a convenient way to enter the needed input data for the algorithm. The callback-method of the dialog was edited to check whether the textboxes are filled when any of the buttons of the dialog is clicked, except the cancel button. The callback-method also starts the algorithm when the *OK* - button is clicked.
- **ReSchedDialog (Derived from Dialog)**  
This dialog gives the user the possibility to select a rescheduling solution, and various options.
- **ResultsDialog (Derived from Dialog)**  
This dialog displays the results of the rescheduling to the the user in a consolidated form. The user may investigate the results further using the results table which is implemented as table.
- **SingleProc (Derived from SingleProc)**  
The processing time was set to “List (Type)”.
- **GanttChart (Derived from GanttChart)**  
Eight entries for layers were added to the options table in order to give the visualized jobs different colours.

## Appendix 5: Job finishing times – Test 5

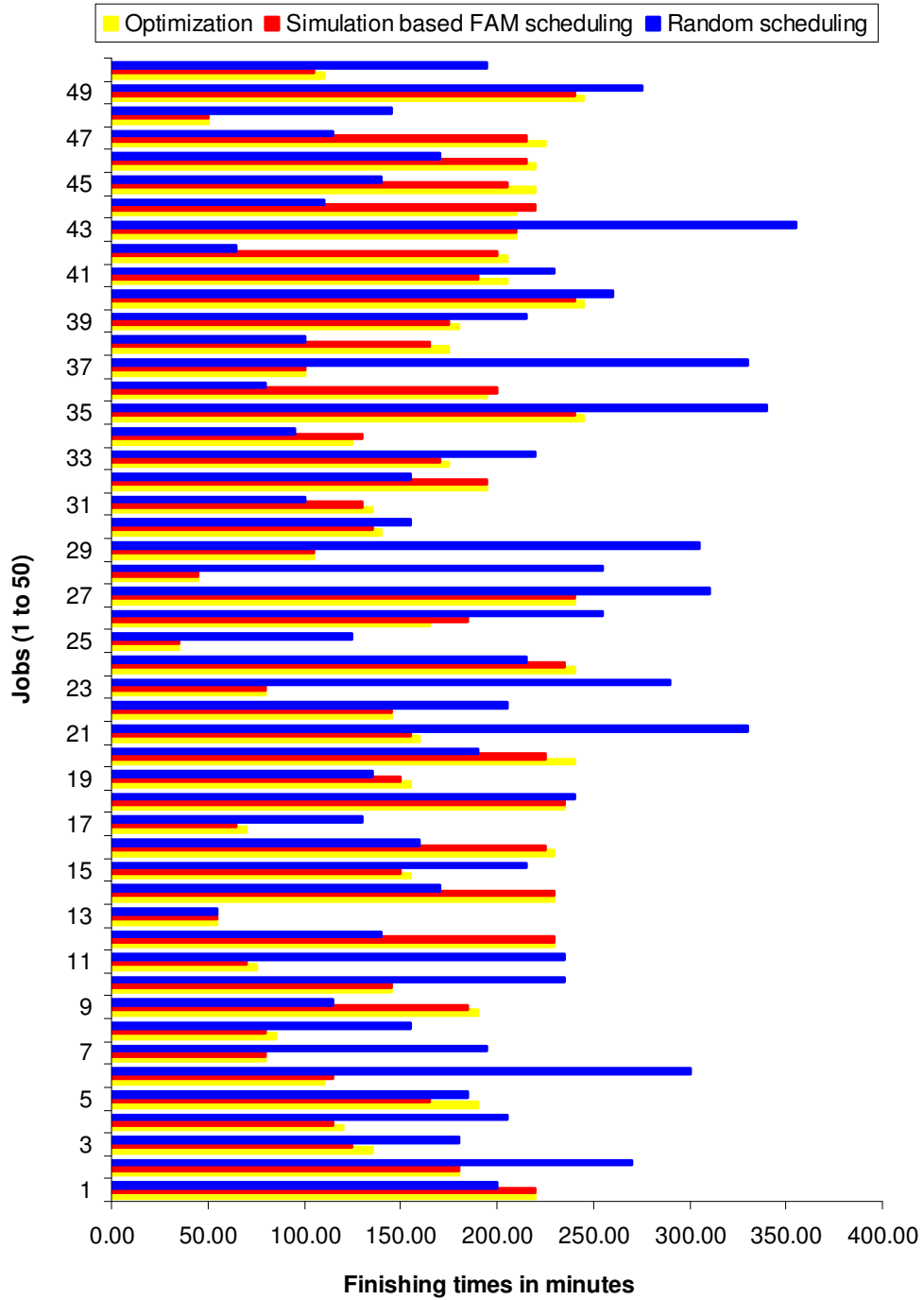


Figure A12: Analysis of job finishing times for different methods

## Appendix 6: Job finishing times – Test 6

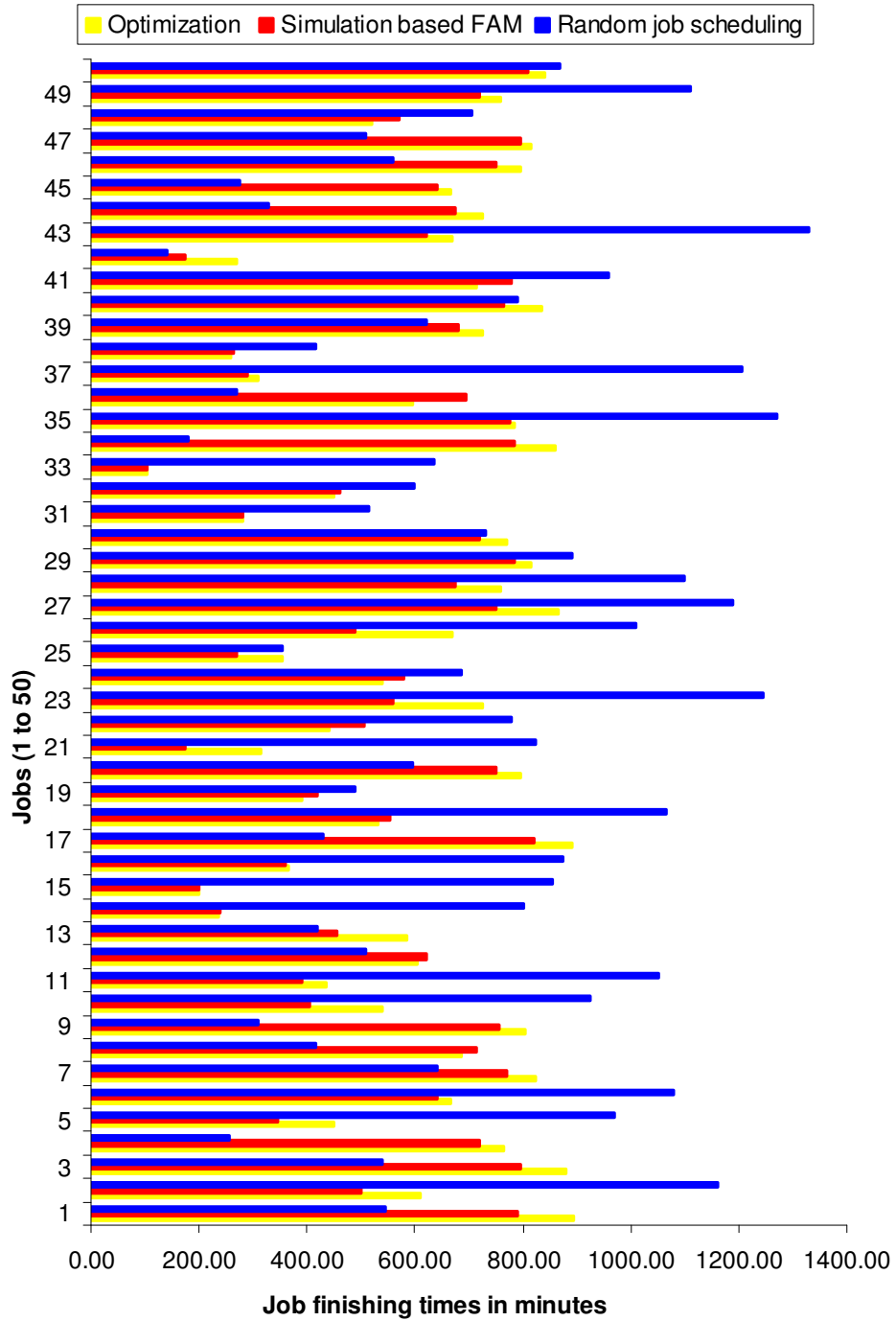


Figure A13 Analyzing job finishing times for different scheduling methods