INTEGRATED DECISION SUPPORT FOR FLEXIBLE MULTIPURPOSE PLANTS

BY

JULIAN GRAHAM RICKARD
M.Eng, A.C.G.I.

March 2000

A thesis submitted for the degree of Doctor of Philosophy of the University of London and for the Diploma of Membership of the Imperial College

Centre for Process Systems Engineering
Department of Chemical Engineering and Chemical Technology
Imperial College of Science, Technology and Medicine
London SW7 2BY, U.K.
Traditionally, the bulk of chemical manufacturing has taken place in large continuous facilities, since the economies of scale and low labour requirements meant that this offered the lowest production costs. Batch plants were used only for products that were difficult or uneconomic to convert to continuous operation. However, reduced margins in bulk chemicals have led to renewed interest in small flexible batch plants capable of producing a number of products. These plants allow a manufacturer to gain a marketing edge by tailoring products to specific customer requirements.

Due to their extremely dynamic nature, these plants are typically difficult to operate close to their maximum capacity, and over the years many attempts have been made to use computer technology to improve the management and control of these plants. Much of this technology is beginning to see industrial use, but less attention has been paid to how various software solutions work together.

This thesis focuses on integrating plant scheduling software that decides which products should be made when, with the supervisory software that manages the execution of that schedule in the plant control system. The approach taken is to build a single data model, and then to use the structure of this model to pass information between the software packages. By focusing on a model, this approach is generic to any plant which can be modelled, and is easily adapted as plant and products change. This offers significant advantages over approaches focusing on mapping, which are generally bespoke and inflexible, or to approaches connecting the various systems into a common database.

Once information can be freely passed between applications, the triggers for off-line rescheduling, in order to reoptimise the schedule when conditions change from those under which the initial schedule was created, can be studied, and a number of alternative decision primitives are proposed.

An implementation of the model and associated algorithms is demonstrated on a number of examples, highlighting the consistency and robustness of the approach.
Firstly I must thank my two supervisors, Professor Sandro Macchietto and Dr Nilay Shah, for both their help and guidance over these three years, and also for giving me the freedom to pursue many of my own ideas.

Several personnel in Air Products and Chemicals Inc. have been excellent in both providing information for the case study and also in providing industrial perspectives and interest in the work. These include Kim Kerr, Doug Skoyles, Art Garcia, Phil Lyman, Martin Friedl and Vince Grassi. In addition several useful conversations were had with Chris Jakeman of Process Systems Enterprise Ltd.

Of particular help on the computing front, Terrence Crombie has been superb in helping me with interfacing to the SUPERBATCH software.

The current fashion is not to mention people in the centre by name for fear of leaving someone out. My three years here have been a lot of fun and much of that is due to the friendly atmosphere. Thanks to all those who helped with technical problems, the disasters of life and inane discussions over lunch.

Finally I should knowledge the members of Imperial College Mountaineering Club, Epsom Cycling Club and St Jude's Church, who have helped to stretch me physically and spiritually while my PhD has hopefully been stretching me mentally.
## Contents

1 Introduction .................................................. 12
   1.1 The Process Industry ..................................... 12
       1.1.1 Flexible Manufacturing ............................. 13
   1.2 Operation of Process Plants ............................. 14
       1.2.1 Hierarchy of Operations for Flexible Manufacturing 15
           1.2.1.1 Enterprise-Wide Planning .................... 16
           1.2.1.2 Short-Term Scheduling ...................... 16
           1.2.1.3 Supervisory Control ....................... 17
           1.2.1.4 Detailed Process Control ................. 17
       1.2.2 Integrated Plant Operations ....................... 17
           1.2.2.1 Data Flow .................................. 18
           1.2.2.2 Information Management ..................... 20
           1.2.2.3 Model Management .......................... 20
       1.2.3 International Standards in Batch Control and Enterprise Integration 20
   1.3 Thesis Outline ........................................... 21

2 Literature Review ............................................. 22
   2.1 Enterprise-Wide Long-term Planning ........................ 22
   2.2 Short-Term Scheduling .................................... 23
       2.2.1 Mathematical Programming ........................ 23
       2.2.2 Heuristics ....................................... 27
       2.2.3 Other Approaches ................................ 28
       2.2.4 Uncertainty in scheduling ........................ 29
   2.3 Supervisory Control ..................................... 30
       2.3.1 Rescheduling and On-line Scheduling ............ 31
   2.4 Integrated Operations .................................... 33
   2.5 International Standards ................................. 36
   2.6 Discussion .............................................. 39
       2.6.1 Information Requirements ........................ 40

3 Model Structure and Definition ................................ 41
   3.1 Model Overview .......................................... 41
   3.2 Equipment ............................................... 41
       3.2.1 Continuous Equipment ............................. 45
Contents.

3.2.2 Common Resources ............................................. 46
3.2.3 Equipment Flowsheet .......................................... 46
3.3 Materials .......................................................... 46
3.4 Recipes and Procedures .......................................... 48
3.4.1 Limited Connectivity .......................................... 52
3.5 Additional Information ........................................... 53
3.5.1 Initial inventory and order slate ............................ 56
3.6 Scope and range of the model .................................... 57

4 Submodel Generation ................................................. 59
  4.1 Information Post Processing .................................... 59
    4.1.1 Defining the suitability of each unit ..................... 59
      4.1.1.1 Modifying Equipment Suitability with Equipment State . 60
      4.1.1.2 Generation of task level details from step data .... 61
      4.1.1.3 Find minimum and maximum sizes for each task ...... 65
      4.1.1.4 Example of generating task details .................. 67
    4.1.2 Further Processing .......................................... 68
      4.1.2.1 Addition of supplier and customer resources ....... 68
      4.1.2.2 Addition of breakdown recipes ....................... 69
      4.1.2.3 Creation of unit operations ......................... 69
      4.1.2.4 Handling stable states .............................. 70
  4.2 Generation of Scheduling Models .............................. 70
    4.2.1 gBSS Model ................................................ 70
    4.2.2 Minimax Scheduling Model .............................. 71
    4.2.3 SA/GA Scheduling Model .............................. 71
  4.3 Generation of the Supervisory Control Model ................ 72
  4.4 Generation of the Plant Simulation Model .................... 72
    4.4.1 Historical 'play-back' facility ......................... 73
  4.5 Conclusions on model generation ............................. 75

5 Implementation of Off-line Schedules .......................... 77
  5.1 Sorting the gBSS schedule ................................... 77
  5.2 Creating a list of control recipes .......................... 77
  5.3 Ordering the list of control recipes ......................... 79
    5.3.1 Translation of deliveries .............................. 81
  5.4 Specifying unit and material allocation in the control recipes .... 81
  5.5 Example of schedule translation ............................ 82
  5.6 Example of schedule reordering ............................. 86
  5.7 Translation from Minimax and SA/GA packages ................ 89
  5.8 Conclusions from schedule translation ...................... 90

6 Rescheduling with the Current Plant Status .................... 91
  6.1 Reading the current on-line schedule ....................... 91
  6.2 The timing of rescheduling .................................. 94
  6.3 Fixing Control Recipes ....................................... 95
    6.3.1 Making equipment unavailable ......................... 95
    6.3.2 Reducing common resource availability ................ 97
    6.3.3 Adding 'virtual' receipts and deliveries ............ 97
Contents.

6.4 Setting initial amounts ........................................ 98
6.5 Preparing the mass balance .................................... 98
6.6 An example of rescheduling with a fixed recipe ............. 98
6.7 Testing the consistency of rescheduling ...................... 100
6.8 Conclusions from rescheduling ............................... 101

7 Prototype Implementation of the System .......................... 102
7.1 Operating the prototype system ............................... 103
7.1.1 Data entry ............................................. 103
7.1.2 Building and connecting the extended system ............. 103
7.1.3 Controlling Information flow ............................. 106
    7.1.3.1 Manual mode ..................................... 106
    7.1.3.2 Automatic Mode .................................. 107
7.2 Schedule Information Handling ................................ 107
    7.2.1 Initial Start-Up ..................................... 108
    7.2.2 Closed Loop Operation ................................ 109
7.3 Interface with SUPERBATCH ................................. 111
7.4 Enhancements to the prototype system ....................... 111
7.5 Conclusions for the prototype implementation ............. 112

8 Closed Loop Operation and Performance ......................... 113
8.1 Rescheduling decision primitives ........................... 113
    8.1.1 Decay in the objective function ...................... 113
    8.1.2 Every x minutes .................................... 115
    8.1.3 New order or receipt notified ....................... 115
    8.1.4 New order or receipt falls within the scheduling horizon .... 115
    8.1.5 Order delayed beyond the due date .................... 116
    8.1.6 Disruption coefficient ............................. 117
    8.1.7 Equipment breakdown ................................ 117
8.2 Updating the on-line schedule ................................ 118
8.3 A performance measure for an integrated system ............ 120
8.4 Interface to a planning system ............................. 120
8.5 Conclusions on system operation and performance .......... 122

9 General Case Studies ............................................. 124
9.1 The Batch Pilot Plant ........................................ 124
    9.1.1 Batch Pilot Plant Results .......................... 127
    9.1.2 Batch Pilot Plant Conclusions ....................... 132
        9.1.2.1 Dealing with major upsets .................... 133
9.2 The BATCIME Reference Plant ................................ 133
    9.2.1 BATCIME Reference Plant Results .................... 138
    9.2.2 BATCIME Reference Plant Conclusions ............... 140

10 Air Products Case Study ......................................... 143
10.1 Plant and Process Description ............................ 143
10.2 Approach to the Case Study ................................ 146
10.3 Detailed Methodology ...................................... 150
    10.3.1 Model generation .................................. 150
    10.3.2 Translation of the coarse gBSS schedule ............ 150
# List of Figures

1.1 Operations hierarchy ........................................... 15  
1.2 Example of a Gantt chart ......................................... 16  
1.3 Methods for integration ......................................... 19  

2.1 Example of a State-Task Network (Kondili et al., 1993). ........ 24  
2.2 Illustration of discrete time representation ....................... 25  
2.3 Illustration of continuous time representation .................... 26  
2.4 Example of a Resource-Task Network ............................ 26  
2.5 Architecture of an advanced operations support system (Liu, 1995). . 34  
2.6 The S88.01 Standard model structure ............................ 37  

3.1 Structure of equipment items in the unified model ................. 42  
3.2 Example of a route critical valve ................................ 43  
3.3 Example flowsheet ............................................. 43  
3.4 Structure of materials in the unified model ....................... 47  
3.5 Structure of recipes in the unified model ........................ 48  
3.6 Graphical representation of tasks and steps ....................... 49  
3.7 Modelling transfers in a discrete time representation .............. 50  
3.8 Modelling transfers with stable states ........................... 51  
3.9 Example flowsheet of a plant with limited connectivity .......... 54  
3.10 State Task Networks for example plants with and without limited connectivity ................................. 54  
3.11 Overall structure of the unified model .......................... 58  

4.1 Various unit holdup profiles ..................................... 66  
4.2 Corresponding STN proportions ................................... 66  
4.3 External interactions to and from the site ......................... 68  
4.4 Interactions between SUPERBATCH, ControlLink and gPROMS ......... 72  

5.1 Finding other tasks associated with a task in a control recipe .... 80  
5.2 Flowsheet of the Example Plant ................................ 83  
5.3 State Task Network of the Example Plant ......................... 83  
5.4 gBSS Gantt chart for the Example Plant .......................... 84  
5.5 SUPERBATCH Gantt chart for the Example Plant .................. 86  
5.6 Example plant and process where reordering may be necessary .... 86
List of Figures.

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.7</td>
<td>gBSS Gantt chart for the Example Plant</td>
<td>87</td>
</tr>
<tr>
<td>5.8</td>
<td>SUPERBATCH Gantt chart for the Example Plant</td>
<td>89</td>
</tr>
<tr>
<td>6.1</td>
<td>Example of RUAPS</td>
<td>93</td>
</tr>
<tr>
<td>6.2</td>
<td>Potential pitfalls of instantaneous transfers</td>
<td>94</td>
</tr>
<tr>
<td>6.3</td>
<td>Initial detailed on-line schedule</td>
<td>99</td>
</tr>
<tr>
<td>6.4</td>
<td>Resulting initial position in gBSS</td>
<td>99</td>
</tr>
<tr>
<td>7.1</td>
<td>Prototype 'front end' on an integrated decision support system</td>
<td>102</td>
</tr>
<tr>
<td>7.2</td>
<td>Application interactions in the integrated environment</td>
<td>104</td>
</tr>
<tr>
<td>7.3</td>
<td>SUPERBATCH on-line Gantt chart</td>
<td>105</td>
</tr>
<tr>
<td>8.1</td>
<td>Typical Objective Function Plot</td>
<td>114</td>
</tr>
<tr>
<td>8.2</td>
<td>Order entry in gBOSS+</td>
<td>116</td>
</tr>
<tr>
<td>8.3</td>
<td>Adding a breakdown in gBOSS+</td>
<td>118</td>
</tr>
<tr>
<td>8.4</td>
<td>Display of control recipe lists</td>
<td>119</td>
</tr>
<tr>
<td>8.5</td>
<td>Display of performance measures</td>
<td>121</td>
</tr>
<tr>
<td>9.1</td>
<td>Flowsheet of the Pilot Plant</td>
<td>125</td>
</tr>
<tr>
<td>9.2</td>
<td>State Task Networks for the Pilot Plant Recipes</td>
<td>126</td>
</tr>
<tr>
<td>9.3</td>
<td>A gBSS Gantt chart for the pilot plant after 28 hours</td>
<td>130</td>
</tr>
<tr>
<td>9.4</td>
<td>The resulting SUPERBATCH Gantt chart after translation</td>
<td>130</td>
</tr>
<tr>
<td>9.5</td>
<td>Flowsheet of the BATCIME Plant</td>
<td>134</td>
</tr>
<tr>
<td>9.6</td>
<td>Full STN for the BATCIME Plant</td>
<td>135</td>
</tr>
<tr>
<td>9.7</td>
<td>Condensed STN for the BATCIME Plant</td>
<td>136</td>
</tr>
<tr>
<td>9.8</td>
<td>Initial gBSS Gantt chart for the BATCIME plant</td>
<td>139</td>
</tr>
<tr>
<td>9.9</td>
<td>Initial SUPERBATCH Gantt chart for the BATCIME plant</td>
<td>139</td>
</tr>
<tr>
<td>9.10</td>
<td>gBSS steam profile for the BATCIME plant with an extra order</td>
<td>140</td>
</tr>
<tr>
<td>9.11</td>
<td>gBSS schedule for the BATCIME plant with an extra order</td>
<td>141</td>
</tr>
<tr>
<td>9.12</td>
<td>SUPERBATCH schedule for the BATCIME plant with an extra order</td>
<td>141</td>
</tr>
<tr>
<td>9.13</td>
<td>gBSS schedule for the BATCIME plant with a unit out of action</td>
<td>142</td>
</tr>
<tr>
<td>9.14</td>
<td>SUPERBATCH schedule for the BATCIME plant with a unit out of action</td>
<td>142</td>
</tr>
<tr>
<td>10.1</td>
<td>Flowsheet of the Clayton Plant</td>
<td>143</td>
</tr>
<tr>
<td>10.2</td>
<td>Scheduling two batches when the drumming line is not switched</td>
<td>146</td>
</tr>
<tr>
<td>10.3</td>
<td>Scheduling two batches when the drumming line is switched</td>
<td>147</td>
</tr>
<tr>
<td>10.4</td>
<td>Overview of the approach to the Air Products Case Study</td>
<td>148</td>
</tr>
<tr>
<td>10.5</td>
<td>Models used at each level for the Air Products Case Study</td>
<td>149</td>
</tr>
<tr>
<td>10.6</td>
<td>Example of reversed detailed targets</td>
<td>152</td>
</tr>
<tr>
<td>10.7</td>
<td>Example gBSS Gantt chart from the coarse model</td>
<td>155</td>
</tr>
<tr>
<td>10.8</td>
<td>Example SUPERBATCH Gantt chart from the coarse model</td>
<td>156</td>
</tr>
<tr>
<td>10.9</td>
<td>Example gBSS Gantt chart from the detailed model</td>
<td>158</td>
</tr>
<tr>
<td>10.10</td>
<td>List of tasks from the detailed model</td>
<td>158</td>
</tr>
<tr>
<td>10.11</td>
<td>Example SUPERBATCH Gantt chart from the detailed model</td>
<td>159</td>
</tr>
<tr>
<td>A.1</td>
<td>Example of a simple STN</td>
<td>175</td>
</tr>
<tr>
<td>C.1</td>
<td>Flowsheet for Charlotte's Plant</td>
<td>185</td>
</tr>
<tr>
<td>C.2</td>
<td>STN for Charlotte's Plant</td>
<td>185</td>
</tr>
<tr>
<td>C.3</td>
<td>Flowsheet for Emma's Plant</td>
<td>186</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>C.4</td>
<td>STN for Emma’s Plant</td>
<td>186</td>
</tr>
<tr>
<td>C.5</td>
<td>Flowsheet for Felicity’s Plant</td>
<td>187</td>
</tr>
<tr>
<td>C.6</td>
<td>STN for Felicity’s Plant</td>
<td>187</td>
</tr>
<tr>
<td>C.7</td>
<td>Flowsheet for Ingrid’s Plant</td>
<td>188</td>
</tr>
<tr>
<td>C.8</td>
<td>STN for Ingrid’s Plant</td>
<td>188</td>
</tr>
<tr>
<td>C.9</td>
<td>Flowsheet for Isabel’s Plant</td>
<td>189</td>
</tr>
<tr>
<td>C.10</td>
<td>STN for Isabel’s Plant</td>
<td>189</td>
</tr>
<tr>
<td>C.11</td>
<td>Flowsheet for APCI Plant (non-confidential version)</td>
<td>190</td>
</tr>
<tr>
<td>C.12</td>
<td>Flowsheet for the Batch Pilot Plant</td>
<td>191</td>
</tr>
<tr>
<td>C.13</td>
<td>STN for the Batch Pilot Plant</td>
<td>191</td>
</tr>
<tr>
<td>C.14</td>
<td>Flowsheet for Mel’s Plant</td>
<td>192</td>
</tr>
<tr>
<td>C.15</td>
<td>STN for Mel’s Plant</td>
<td>192</td>
</tr>
<tr>
<td>C.16</td>
<td>Flowsheet for Nancy’s Plant</td>
<td>193</td>
</tr>
<tr>
<td>C.17</td>
<td>STN for Nancy’s Plant</td>
<td>193</td>
</tr>
<tr>
<td>C.18</td>
<td>Flowsheet for Patricia’s Plant</td>
<td>194</td>
</tr>
<tr>
<td>C.19</td>
<td>STN for Patricia’s Plant</td>
<td>195</td>
</tr>
<tr>
<td>C.20</td>
<td>Flowsheet for Paula’s Plant</td>
<td>196</td>
</tr>
<tr>
<td>C.21</td>
<td>STN for Paula’s Plant</td>
<td>196</td>
</tr>
<tr>
<td>C.22</td>
<td>Flowsheet for Ruth’s Plant</td>
<td>197</td>
</tr>
<tr>
<td>C.23</td>
<td>STN for Ruth’s Plant</td>
<td>197</td>
</tr>
<tr>
<td>C.24</td>
<td>Flowsheet for Sarah’s Plant</td>
<td>198</td>
</tr>
<tr>
<td>C.25</td>
<td>STN for Sarah’s Plant</td>
<td>198</td>
</tr>
<tr>
<td>C.26</td>
<td>Flowsheet for Tracy’s Plant</td>
<td>199</td>
</tr>
<tr>
<td>C.27</td>
<td>STN for Tracy’s Plant</td>
<td>199</td>
</tr>
<tr>
<td>C.28</td>
<td>Flowsheet for Carlene’s Plant</td>
<td>200</td>
</tr>
<tr>
<td>C.29</td>
<td>STN for Carlene’s Plant</td>
<td>200</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Features of Batch Processes ........................................... 42
3.2 Definition language for equipment .................................... 44
3.3 Definition language for continuous equipment .................. 46
3.4 Definition language for common resources ....................... 46
3.5 Definition language for equipment flowsheet ................... 46
3.6 Definition language for materials ..................................... 47
3.7 Definition language for operations ..................................... 48
3.8 Definition language for master recipes ............................... 50
3.9 Definition language for steps in master recipes ................. 53
3.10 Definition language for steps with limited connectivity ....... 55
3.11 Definition language for additional information .................. 55
3.12 Definition language for initial state and order information .... 57

4.1 Example gPROMS model language for a unit ..................... 74
4.2 Example gPROMS model language for a unit operation .......... 74
4.3 Example gPROMS model language for the monitor task .......... 75
4.4 Format of file for specifying historical data ..................... 75

5.1 State Task Network for the example in tabular form ............. 84
5.2 Chronological list of tasks for the Example Plant ............... 85
5.3 Initial order of control recipes ...................................... 88
5.4 Reordered list of control recipes ................................... 88

6.1 Action taken on each column in the RUAPS file .................. 93

7.1 Initial Schedules .................................................. 108
7.2 Schedules after initial gBSS run .................................. 109
7.3 Schedules after promoting wannabe schedule to pretender .... 109
7.4 Schedules after promoting pretender schedule to actual ........ 109
7.5 Schedules after starting rescheduling in gBSS ................. 110
7.6 Schedules after reading the results of rescheduling ........... 110
7.7 Schedules after rescheduling ...................................... 110

8.1 Format of the planning level interface ............................ 122

9.1 Order slate for the case study .................................... 127
List of Tables.

9.2 Value added for various scheduling horizons and update intervals. (Backwards translation as described in Algorithm 5.1) .......................... 128
9.3 Value added for various scheduling horizons and update intervals. (Forwards translation as described in Algorithm 5.2) .................. 128
9.4 Value added for various percentages of variation in the model. .......... 131
9.5 Value added for various values of the rounding coefficient. ........ 131
9.6 Value added for various inviolate period lengths. .................. 132
9.7 Value added for various rescheduling policies. (10% variation) .......... 132
9.8 Resources and Units for the BATCIME reference plant. .............. 137
9.9 Composition of the various products. .................................. 138
9.10 Initial BATCIME order slate. ........................................ 138
10.1 Order slate for APCI Case Study. .................................. 144
10.2 Units in the Case Study. ........................................ 144
10.3 Cleaning Matrix for Reactor 1. .................................. 144
10.4 Cleaning Matrix for Reactor 2. .................................. 144
10.5 Materials in the Air Products Case Study. .......................... 145
10.6 Cleaning Matrix for Reactor 3. .................................. 146
10.7 Cleaning Matrix for Reactor 4. .................................. 146
10.8 APCI results when loop update timings are changed. .................. 160
10.9 APCI results when rounding is changed. ................................ 161
10.10 APCI results when model variability is varied. ...................... 162
A.1 Definition language for states and tasks in gBSS. ..................... 176
A.2 Definition language for continuous tasks in gBSS. ...................... 176
A.3 Definition language for equipment in gBSS. .......................... 177
A.4 Definition language for the problem statement in gBSS. .............. 177
B.1 Definition language for equipment in SUPERBATCH. ................. 179
B.2 Definition language for common resources in SUPERBATCH. .......... 180
B.3 Definition language for flowsheets in SUPERBATCH. .................. 180
B.4 Definition language for materials in SUPERBATCH. ................... 180
B.5 Definition language for phases in SUPERBATCH. ..................... 181
B.6 Definition language for recipes in SUPERBATCH. ..................... 182
B.7 Definition language for initial inventories in SUPERBATCH. ........ 182
B.8 Definition language for scheduling batches in SUPERBATCH. ......... 183
C.1 Features of Batch Processing which can be modelled. ................. 184
This thesis describes a methodology for enabling the automatic transfer of information between various decision making functions in the operation of process plants. To start with we look at the process industry in which this takes place.

1.1 The Process Industry

The process industry is that sector of industry in which materials are changed by physical or chemical processes. This covers everything from power generation, bulk material processing such as steel making and oil refining to pharmaceuticals manufacturing and food processing. The processes by which these changes take place can be divided into continuous processes, where raw materials are continuously fed and products drawn, and batch processes, where product is made in discrete amounts.

Before the industrial revolution, nearly all production was by batch processes, for example iron smelting. Processes were highly labour intensive, but then there was no shortage of labour. The advent of steam power and pumping technology allowed much larger and more complex processes to be operated. The equipment and plants became more expensive and were therefore operated around the clock, and were designed to produce continuous flows of products.

However in order to keep these plants operating near their desired conditions, manual intervention was required to adjust flows etc. in order to keep parameters of the processes such as temperatures near their required levels.

Rising labour costs, the development of electronics and breakthroughs in control technology, e.g. Ziegler and Nichols (1942), saw that automatic controllers replaced manual control. This is epitomised by the modern oil refinery, where huge quantities of crude oil undergo a vast range of processes and the whole site, covering many acres, can be operated by a hand-full of staff.
While continuous processes dominated, the batch mode of operation remained dominant for certain classes of processes. These include processes requiring large amounts of cleaning, such as food processing, highly regulated processes such as pharmaceuticals where raw material identification is important, lengthy processes such as brewing, and very small scale operations such as perfume manufacture. These are typically high value added products, and therefore the labour costs involved are less important than with bulk products.

With labour costs all but removed from production costs, plants have become larger and larger as companies seek to benefit from economies of scale, and the prices of bulk chemicals have fallen as companies seek to undercut each other. Overcapacity in these markets means that profit margins and potential for growth are low, and companies are at the whim of external factors such as monetary exchange rates, suffering cycles of large profits and losses.

This intense competition, most notably from Asia, the Middle East and North America, in commodity chemicals, for example polyethylene and PVC, has led many companies to concentrate on low volume, high value-added products (Economist, 1997), such as speciality and fine chemicals. In these products, a marketing edge can often be attained by adapting the product to an individual customer's needs, products are often made to order rather than made for stock, and the concepts of Just In Time, (JIT), and On-Time In-Full, (OTIF), have taken root in the process industries (Benson, 1995).

1.1.1 Flexible Manufacturing

A key advantage of batch plants is that they are dynamic and flexible, since there is no reason why the same product must be made every batch, and this gives rise to multi-product plants. The products may still all be fairly similar, but there is the possibility of tailoring a product recipe specifically for a particular customer. For example a customer may be willing to pay a premium if they can get a product at a higher concentration for use in their process and this might be achieved simply by adjusting the amount of water added in a recipe. A batch plant is more likely to be able to cope with decreases in transfer rates for the potentially more viscous product than a continuous plant where the effect will be felt throughout the process (Rippin, 1991).

Where there are many different equipment items it may be possible to manufacture a product in various ways using different combinations of the equipment. Each equipment item may be able to perform one or more tasks. For example one vessel may be used as a mixer or a reactor, while another might only be a mixer, since it lacks any temperature control. Therefore there is a choice in what equipment is used for any particular batch. This is a multi-purpose plant.
Both of these represent an increase in flexibility in the way they may be operated over plants where each piece of equipment performs a single task. This flexibility can be used to match more closely the quantity of products produced to the demand, and therefore the capital expenditure on equipment can achieve a greater return. There is also the ability to quickly increase production of a particular product should a new market appear.

It is also possible for continuous plants to manufacture a number of products, although these tend to be very similar in nature. The ideas and techniques described in this thesis are developed for batch multipurpose plants, but most could easily also apply to these multi-product continuous plants.

1.2 Operation of Process Plants

If the aim of a company is to satisfy its customers, then the aim of operating a plant producing product is to satisfy the customers demands for product “on time, in full”, at minimum cost to the producing company. Therefore the plant should not be any bigger than it has to be, and neither should the stocks of product be any larger than necessary.

In a continuous plant manufacturing a single product, the only operational requirements are for the regulation of plant parameters to keep the operation within the desired limits, and possibly adjusting the overall production rate in response to changing market demands. In a batch plant manufacturing a single product there are now a number of discrete events at each point in the process where there is a change in the operation. However if the operation consists of repeating the same batch then there is usually a clearly defined sequence of events and therefore it is simple for human operators to follow this cyclic sequence.

However in a flexible multipurpose plant there is no clear cycle of events, and a host of questions from “what products should be made this month?” to “which reactor should be charged next with raw materials?” must be answered in order to run the plant. Leaving all these questions to human operators is likely to leave a large margin for improvement.

In the modern competitive environment the drive towards better management of these plants is often necessary for the survival of a business. Lemetais (1999) states that the key to profitability in these flexible plants is managing production of ever smaller batches required on ever shorter lead times. Therefore some way of handling the huge amount of information thrown up by large numbers of small batches, and in quickly searching through the alternatives in how best to operate the plant in response to sudden changes in requirements must be found.

Computer technology is often applied to situations where there is a vast range of possible alternatives or a large amount of rapidly changing information, and the operation of flexible process plants has both these features. However, even with computers, the entire
problem of how to operate a complex plant is too large to be tackled as a single problem. Therefore the problem has been broken down into a number of smaller problems, which fit into a hierarchy, discussed in the next section. The use of multiple computer systems leads to the concept of Computer Integrated Manufacturing, CIM, where all aspects of manufacturing, from planning and control to materials procurement and customer order planning are tied together.

1.2.1 Hierarchy of Operations for Flexible Manufacturing

The hierarchy breaks the operations problem down into a number of smaller problems, each with a different time scale. At each level the equipment and processes may be represented with a different level of detail, taking into account what is useful given the time scale.

![Operations hierarchy diagram]

Figure 1.1: Operations hierarchy.

Figure 1.1 shows an example of such a hierarchy. Different authors and organisations vary over the names and shapes of such a diagram, e.g. Bassett et al. (1996), ISA (1995) and Puigjaner and Espuña (1998), but the overall structure and breakdown of the problem is the same.

The aim of the hierarchy is to break the problem down into manageable parts, while retaining the feature that decisions made at a particular level may have an impact on the others. The various layers in this hierarchy are discussed in turn.
1.2.1.1 Enterprise-Wide Planning

At the top of the hierarchy is a planning function, where decisions are made across the entire enterprise, which may consist of a number of processing sites, perhaps in different countries, over a time-span of months, possibly years.

The aim at this level is to allocate the production of products between the various sites. The result might be weekly targets for the production at each site, or sets of orders that a particular site will be responsible for fulfilling.

1.2.1.2 Short-Term Scheduling

The targets for a particular site must now be turned into a list of batches that will be executed. The decisions to be made for the period in question, typically around a week, include:

- How many batches of each product should be made?
- What size should each batch be?
- What equipment should each batch use?
- What sequence should these batches be made in?

The aim is typically to meet as much of the targets as possible while minimizing costs such as cleaning or changeovers that may be required when switching between products. The result is typically shown as a Gantt chart, where the major processing units are listed along one axis, and their actions then marked off over time along the other axis, as shown in Figure 1.2.

![Gantt chart example](image)

Figure 1.2: Example of a Gantt chart.
1.2.1.3 Supervisory Control

The list of batches obtained at the short-term scheduling level must be executed in the plant. This involves the resolution of every real-time conflict for every resource, such as exactly which transfer will use a pipeline first. Therefore, a detailed view of the process and equipment on a minute by minute basis is required. This view must capture all possible conflicts for resources, some of which may have been ignored at the short-term scheduling level since they have only a minor impact on plant operations over the course of a week, but must nevertheless be resolved for the plant to operate.

At the supervisory control level, the various steps of each recipe are initiated in the detailed control system (e.g. a Programmable Logic Controller, PLC, or Distributed Control System, DCS), and the completion of each step is responded to. Where software is developed to perform this, it is sometimes referred to as a Computer-Aided Production Management, CAPM, system.

1.2.1.4 Detailed Process Control

The individual steps managed at the supervisory control level may consist of a large number of detailed valve movements, equipment switches etc. This sequential control, together with the regulatory control of maintaining flow rates or temperatures in the process, is outside the scope of this thesis.

1.2.2 Integrated Plant Operations

Each of these levels in the hierarchy is often studied as a problem in its own right. However several authors have highlighted the need to consider the interactions between different planning levels, e.g. Puigjaner and Espuna (1998) and Bassett et al. (1996).

None of these single levels leads to optimal plant operation by itself, no matter how well it solves its own part of the problem, and each is reliant upon information generated by other levels in the hierarchy. If each level represents a computer application then this information ought to flow automatically between applications, without the intervention of any operator. This frees staff from straightforward data transfer and manipulation, and allows them to investigate alternative plans, and explore other avenues for improved plant operation.

Two of the features cited by Zentner et al. (1994) as making the practical solution of scheduling problems difficult are the dynamic nature of the manufacturing environment and the quantity of information present. If the scheduling solution exists in isolation, then the large quantity of rapidly changing information will have to be provided manually. However if data is passed automatically then staff can concentrate on the overall picture, rather than on mundane tasks, which are additionally prone to error if carried
out manually. In addition, when upsets from normal operation do occur in a plant, for example a pump breaks, often all available manpower is thrown into fixing the problem, and little attention is given to managing plant operation around this problem.

Looking at the short-term scheduling and supervisory control functions, the aim of an integrated system is for the results from scheduling to be immediately and automatically available to the supervisory control system, and for information about the current plant activities to be available to the scheduling system. Without such a link, a scheduling system is detached from the business process and is not being used as effectively as it might be.

However the task of passing information between the levels is not often considered and when it is it is usually left to someone else. For example, Schilling (1997) describes advanced rescheduling algorithms which might form part of a Computer-Aided Production Management system, but expects that system to be responsible for passing all necessary data in the correct format.

The root of this problem is that the model used at a particular level does not contain the information required at other layers, and therefore communication between levels is hindered. For example the supervisory system might have a model of recipes made up of detailed steps, but it does not know anything about the model used by the scheduling package.

When thinking about an integrated system, another consideration is the response of the integrated system to changes in the plant and process that is being operated. Flexible manufacturing offers great advantages in markets where product lifespans are short and therefore the systems used to operate the plant must also be able to be changed quickly and accurately with the introduction of new product recipes or modification to the equipment. This is why individual applications are usually model based, so that they are easily adaptable, and may be applied to a wide range of situations.

Stanley (1994) describes three paradigms for integration which will be described in turn. The overall views are shown in Figure 1.3.

1.2.2.1 Data Flow

For given models the mapping between them may be defined and this mapping may be used to translate results from one package into another.

Such an approach may be modified to suit just about any situation, but once created, the mapping must be updated whenever the products or equipment are changed, and therefore tends to be inflexible. The effort required to write the mapping routines is fairly trivial when there is one to one mapping between the models, but as the models become more different the routines become more complicated and harder to write.
Figure 1.3: Methods for integration.
Chapter 1. Introduction

There is also no guarantee of consistency in the models used since they have been created independently and the information that is flowing is subject to any faults in the mapping routines.

1.2.2.2 Information Management

This approach rests upon a single common database where all the required information is stored. The various applications read and write data to and from this database, thereby ensuring that the data is consistent. Again there is no guarantee that the models used are consistent. The method also requires that the models are very close to identical, at least where they share information, since the data is not manipulated in any way.

1.2.2.3 Model Management

The previous two approaches have focused on data flow, which allows redundant, possibly inconsistent model information to be encoded in multiple applications, complicating the development and maintenance of integrated systems.

If there is a single common data model from which all application models are generated then both data and model consistency are ensured. Therefore use is made not only of common information between applications, but also of common representations of equipment and procedures. For example, if there is a reactor in the plant, both the scheduling and supervisory models will contain this item, together with information about it. If there is no link between the models then there is no guarantee that the information is the same, and since it was probably written by different people with different backgrounds at different times, there is some potential for conflicting information.

Updating a system based upon a model is also simple, since by updating the single model, the sub-models may be updated in a consistent fashion.

However this approach requires a general model that contains all the information required for the sub-models, in such a structure that allows information to be passed between different applications and also makes logical sense to someone trying to construct the model.

There are also some international efforts in creating standards for information representation in this area, which such a model should conform to, or at least take account of, and therefore these are now described.

1.2.3 International Standards in Batch Control and Enterprise Integration

The various problems in plant operation have spawned a vast number of different computer applications, each with its own view of what the problem is and how to solve it. However,
companies are loath to implement these solutions if they cannot work together. Examples of software in batch manufacturing are laboratory information systems, supervisory control systems, warehousing systems, financial systems and historical data repositories.

Therefore several societies, such as the Instrument Society of America (ISA), and groups of companies have begun to try and define international standards in the models used by the various systems to meet, so that they all use the same terminology. The aim is for users to be able to “plug and play” with different applications, reducing the implementation costs and complexity. Any new attempts at systems or models in the area of batch processing must therefore conform to the appropriate standards.

1.3 Thesis Outline

First in Chapter 2, the current state of the art in planning, scheduling and on-line execution are reviewed, together with previous attempts at their integration. The case is then put for developing further methods for integration.

Chapter 3 describes a model and definition language which contains all the data required for both off-line scheduling and on-line execution. Chapters 4, 5 and 6 describe algorithms and methods which may be used with the model to allow information to be passed between applications.

Chapter 7 describes a prototype implementation of the model and algorithms previously described, while Chapter 8 looks at how the system may be run to automatically aid decision making in process operations. Chapter 9 describes some studies on various plants, and Chapter 10 describes an application of this technology, with some specific modifications, to a particular industrial plant.

Finally in Chapter 11 the advances made are summarised along with future directions for research and development.
Chapter 2

Literature Review

Over the past twenty years the literature concerning batch plants and their operation has grown considerably. In this chapter the development and current position in each level of the hierarchy introduced in Section 1.2.1 is reviewed, the work done in integrating these levels examined, and the issues involved in rescheduling, where a current schedule is reformulated during its execution, are studied. Then some current and emerging international standards in these areas are discussed. Finally the possibility for improving upon the current state of integration work is discussed and the direction for the rest of this thesis is laid out.

2.1 Enterprise-Wide Long-term Planning

The planning of production over the long term across possibly a number of manufacturing sites is a large problem which is often solved manually by a central planning group. To some extent it is similar to a huge short-term scheduling problem where detail is less critical, but the time span of interest is much longer. Therefore formal techniques tend to use a very aggregated view of what a plant can produce. For example, early work by Mauderli and Rippin (1979) first calculated the timing of single products, then of campaigns, before using linear programming to construct a plan from the campaigns.

The discrete parts manufacturing industries have adopted Materials Requirements Planning (MRP) software and Manufacturing Resource Planning (MRP II) software to help in the planning process, and these have to some extent been developed for use in the process industries (Sawyer, 1990). However a number of features make the application of such technology more difficult in the process industry. These include the fact that the process industries deal with material which can be split, mixed or blended, and must usually be stored within finite vessels. This represents a considerably more complex problem than discrete parts manufacturing, which may explain why the use of such technology is far more limited in the process industries.
Several recent approaches have looked at breaking the full long-term scheduling problem into a number of subproblems, each of which are of a more manageable size. This has particular benefits in multi-product and multipurpose plants, where the product mix and the way in which equipment is used can have a marked effect on a nominal production rate for a single product (Rippin, 1991). However the problem with this approach is how to reconcile global decision making processes with solving the detailed subproblems.

Wilkinson et al. (1995) present an aggregate formulation, where groups of related variables in the exact formulation are replaced by aggregated variables. The time horizon is broken down into a number of Aggregated Time Periods (ATPs), each described by aggregated variables. This formulation can be solved in considerably less computational time and often gives a tight upper bound on the solution to the original detailed formulation. In Wilkinson et al. (1996) it is shown how this approach can be used to solve a problem involving a number of production sites together with the distribution of products through a number of warehouses.

A key feature of such methods is how to ensure that the overall problem remains feasible when the entire problem is not being solved in detail at the same time. Subrahmanyam et al. (1995), who also present a decomposition strategy which breaks the horizon down, use a diagnostic formulation to check for and resolve any infeasibilities that may arise in the overall solution. Wilkinson et al. (1995) on the other hand uses linking variables to take account of the "end effects" for each ATP.

Dimitriadis et al. (1997) present three "rolling horizon" algorithms based on the rigorous aggregated formulation of Wilkinson et al. (1995). By dividing the horizon up into a number of ATPs, small parts of the schedule are solved in detail and then fixed, while the remainder of the schedule is treated as an aggregate model. This allows very long schedules to be solved to near optimality in a reasonable time.

2.2 Short-Term Scheduling

From some of the earliest work on scheduling onwards the scheduling problem has thrown up a huge number of alternatives (Takamatsu et al., 1979), and the aim has been to reduce this solution space while holding on to the optimal, or at least good, solution(s) relative to some objective.

The methods can be divided into those using mathematical programming, heuristics and random search techniques. These are discussed in turn, with particular attention paid to the information required by these techniques.

2.2.1 Mathematical Programming

Kondili et al. (1993) present the general scheduling problem as a Mixed Integer Linear
Programming (MILP) formulation. This is based around the State Task Network (STN) representation for recipes. An example STN is shown in Figure 2.1. The circles represent material "states", and the rectangles represent "tasks", which transform one set of states into another. Information about the units available and which tasks may be performed in each unit, together with the storage for each state, completes the model. The problem to be solved is posed by specifying the material prices and/or orders, together with any costs for performing tasks or using resources.

![Figure 2.1: Example of a State-Task Network (Kondili et al., 1993).](image)

The resulting mathematical problem can be very large, leading to lengthy solution times, and so special techniques were applied to try and improve the solution times (Shah et al., 1993). Development of this work resulted in the general Batch Scheduling System, or gBSS, software (gBSS User Manual, 1997).

gBSS is a software package which reads a set of input files describing a plant, process and problem in the format described in Appendix A. The model is then written as an MILP and sent to the user's choice of solver, which might be either internal routines within gBSS or a third party package, such as CPLEX (©ILOG 1997-1999) or XPRESS (©Dash Associates 1984-1994). The software then reads the result from the solver and allows the user to examine the schedule either graphically as a Gantt chart or in text files.

The formulation of Kondili et al. (1993) is based on a discrete time representation, as shown in Figure 2.2. The horizon H is equally divided into n periods, and events are assumed to occur only at the points in between these periods. Therefore as the interval size increases the model is likely to move further away from the real situation, since the task durations specified in a discrete time model need to be rounded to exact multiples of the time interval. Ideally, the interval size should be the highest common denominator...
Chapter 2. Literature Review

of the discrete events in the model, (such as task durations, deliveries etc.). However this can give rise to unnecessarily large problems where there are both short and long tasks.

\[ H \]

\[ \begin{array}{cccccc}
1 & 2 & 3 & 4 & \ldots & \ldots n & n+1 \\
\end{array} \]

Figure 2.2: Illustration of discrete time representation.

Close analysis of the underlying MILP model structure has led to further improvements, for example Yee and Shah (1998) describe various methods for tightening the relaxation gap of the formulation which leads to faster solution times. For example the presence of non-productive tasks such as changeovers tend to increase the gap between the optimal solution and the fully relaxed LP, so cut constraints are introduced, reducing the gap and improving the solution time. Such improvements are obtained without any changes in the model obtained from the user.

A large amount of subsequent work on scheduling is also based upon the STN representation. For example Rotstein et al. (1994) integrate knowledge about how the plant is operated into the MILP formulation in order to improve the computational efficiency. By adding various constraints to the original formulation, preferences such as which equipment should be used if possible can be added.

An alternative to general formulations are those which focus on special cases in attempts to reduce their problem size and solution time. For example, Pinto and Grossmann (1997) formulate discrete resource constraints, such as a limited number of operators, as disjunctives and so remove these features from the MILP, and reduced solution times compared to a general formulation are shown for some examples.

Voudouris and Grossmann (1996) present a specialised formulation for multi-product scheduling problems where not all the products use the same routes, and the manufacturing of the products can be characterised through production routes, which must be sequential through the equipment. The formulation is an improvement over a general formulation in terms of problem size and solution effort, but is only applicable to plants and processes with these particular characteristics.

Approaches where the horizon is treated as a number of uniform intervals often appear to have more intervals, and therefore a larger problem size, than is really needed. In an attempt to overcome this, Schilling and Pantelides (1996) present a general formulation based on a continuous time representation, where the horizon is treated as shown in Figure 2.3. The horizon, H, is now divided into m periods, and events are again assumed to only
occur at interval boundaries. However the position of these event boundaries is not fixed, and therefore fewer intervals may be required than in the uniform approach shown in Figure 2.2, i.e. it is hoped that \( m \) can be made much smaller than \( n \). This would lead to a smaller problem size, while the problem is also modelled more closely, since exact task durations may be used. However the determination of the position of the event boundaries is now part of the problem to be solved, and therefore the continuous formulation does not always lead to shorter solution times. Tahmassebi (1997) also presents a continuous time scheduling formulation and shows an example where results are obtained significantly faster than with a discrete time approach.

![Figure 2.3: Illustration of continuous time representation.](image)

Ierapetritou and Floudas (1998) present a continuous time scheduling formulation where unit event and task event timings are decoupled. This results in a smaller formulation and shorter solution times, and remains based on the STN representation. It also allows the task durations to vary with respect to the amount of material being processed by the specific task, while remaining a linear formulation.

Pantelides (1994) describes the Resource Task Network, (RTN), as a more general representation than the STN, and an example of this is shown in Figure 2.4. This representation treats all materials, equipment and common resources equally as resources which may be consumed and produced by tasks. This makes some situations easier to model, for example distribution networks, and leads to a neater mathematical formulation. However the underlying problem size and complexity are unchanged.

![Figure 2.4: Example of a Resource-Task Network.](image)
Another limitation of the formulation of Kondili et al. (1993) is that the processing time of tasks is fixed. However if the processing times vary with batch size the formulation might become non-linear. Zhang and Sargent (1994) propose a non-linear formulation for mixed production facilities containing both batch and continuous operations. However the solution of such a formulation is problematic, since the problem is often non-convex and therefore global solutions are impossible to find with current algorithms, although current work in locating the solutions of non-linear problems may open up new possibilities in the solution of such formulations (Floudas, 1999). Zhang and Sargent (1996) go on to describe extensions and improvements to their original formulation, and describe a treatment of continuous tasks which allows the non-linear problem to be transformed into an MILP.

2.2.2 Heuristics

An alternative scheduling methodology is to attempt to replicate the way in which a human scheduler arrives at a schedule through a number of heuristics, or “rules of thumb”. Human schedulers usually gain experience over a number of years and have their own individual methods for planning plant production, which are then replicated in software.

Solutions are often obtained very quickly and depending on the rules used the method may be fairly robust at finding solutions. However there is no guarantee that the solution is optimal, since there is no search of the solution space. In fact the quality of the solution is highly dependent on the rules chosen, and unlike a human scheduler, a heuristic software package does not know or respond to exceptions which might break the rules.

Kudva et al. (1994) present a fairly general heuristic strategy and for some examples show that the results obtained are fairly close to those obtained by optimal scheduling methods. Often, however, heuristics struggle to cope with more complex scheduling situations and require augmentation. For example Graells et al. (1995) present special techniques for handling intermediate storage and Grau et al. (1995) minimise cleaning and changeovers.

Pinedo (1995) describes a wide range of heuristic strategies, a number of which are repeated in the the literature review of Schilling (1997). For example the weighted shortest processing time first (WSPT) rule, which states that for a set of tasks $i$, the tasks are ordered in decreasing order of $\frac{w_i}{\theta_i}$, where $\theta_i$ denotes the duration of task $i$ and $w_i$ denotes a given weighting. This rule can be proven to be optimal for a single machine, but this is not so where there is more than one machine. This trait of providing optimal solutions in very simple cases but being suboptimal for more complex problems is common to most heuristics.

Overall, heuristic methods offer a fast way of obtaining a schedule, but other than in speed, they are unlikely to improve over a manual schedule, since they are only as good as
the rules used. They offer no chance of finding hidden capacity in a plant that is unused because of the current methods of operation, whereas search-based methods may show possibilities that are otherwise missed.

2.2.3 Other Approaches

In addition to mathematical and heuristic approaches, various stochastic methods based on guiding a random search through the solution space have been developed. One of the most popular of these is simulated annealing, (SA). This is a systematic trial and error procedure based on simulation and is capable of dealing with complex problem structures in which there are no general rules for search improvement.

Xia and Macchietto (1994) present an approach based on minimax algebra, which is often used to model discrete event systems, to model a batch plant. This allows the result of a given schedule to be calculated very quickly, which is key in a trial and error search. However an initial sequence is necessary to start the search, and therefore this approach cannot handle complex product recipes with intermediates, where it is not known beforehand what batches must be made.

Xia and Macchietto (1997) then go on to describe the solution of the MINLP formulation of Zhang and Sargent (1996) for the design and scheduling of batch plants, using an evolutionary algorithm (EA) with SA. Evolutionary or genetic algorithms work by keeping good solutions and combining them in the hope that the "off-spring" will be even better. This stochastic approach to MINLP optimisation, implemented in a package named EASY, has the advantage that it is a global search method, for which no gradient information is required, and therefore avoids problems with the severe non-convexity of the problem. Case studies described show that the solution of the non-linear problem can give a very different solution to that of the linearised problem. It is also shown that the EASY algorithm can sometimes find a solution of a linear problem more quickly than gBSS.

Graells et al. (1998) also describe a SA based scheduling package. This is loosely based on an STN representation, but has a hierarchical element where aggregate process can also be treated as a network of operations. Badell et al. (1998) describe how the SA based approach can include cash flow constraints, ensuring that the cash required for purchasing raw materials is available.

Murakami et al. (1997) describe a repetitive method based on SA where a number of different starting points are used, which reduces the probability of the algorithm being trapped in a bad local optimum. Methods for balancing the number of starting points against the number of schedules examined at each point are presented.

An emerging method is the use of Constraint Satisfaction Techniques, (CST), such as
that described by Huang and Chung (1999). The problem is posed as a set of constraints which must be satisfied and then commercial constraint satisfaction library routines are used to produce a schedule. The constraint based approach is compared with the MILP approach by Das et al. (1999), and it is found that the MILP is generally superior, although it benefits from a much longer development.

Sanmarti et al. (1998) present a branch and bound algorithm based on a schedule-graph representation. This works well with flow-shop type operations with single equipment items for a particular task, otherwise the allocation and timing are treated sequentially.

Other methods in scheduling tend to be a little more unique. For example Dockx et al. (1997) take the pessimistic view that scheduling is too complicated to be automated at present, (despite published cases showing the contrary, such as Orcun et al. (1997) for the paints industry and Orcun et al. (1999) for the production of bakers yeast), and present a system to help a human perform the scheduling. This uses heuristic and stochastic techniques on parts of the schedule under the direction of a human scheduler.

2.2.4 Uncertainty in scheduling

The majority of work on scheduling makes the assumption of a certain plant model, where parameters such as processing times, equipment reliability, process yields and demands are fixed. In reality these features may be uncertain, and a few attempts have been made to incorporate this uncertainty into scheduling algorithms.

One approach followed by Ierapetritou and Pistikopoulos (1996) is to create scenarios containing discrete values for all uncertain variables. The scenarios are then combined into a single formulation whose objective function includes the sum of that of each scenario weighted by its probability of occurrence. However this approach quickly leads to very large formulations for even modest sized problems.

In an attempt to overcome this escalation in problem size Bassett et al. (1997) incorporate uncertainty into a detailed scheduling formulation by means of Monte Carlo sampling. Each uncertain parameter is randomly sampled from its given distribution and then the associated problem solved. This process is repeated a number of times until stopping criteria are met. Trends and distributions in the results can be determined and statistically analysed to give a final result. Since each sampling iteration is independent, parallelisation of the method to run on a number of machines is possible, and thus solution times may be kept close to that of the problem without uncertainty. However this is still very computationally expensive.

Therefore although methods exist for incorporating uncertainty into scheduling models, a large price is paid in computational efficiency, which has already been cited as the major limitation in scheduling technology. Sanmarti et al. (1995) present a heuristic
based system that incorporates a reliability index for each piece of equipment, which may change throughout the scheduling horizon as preventive maintenance is carried out. In common with most heuristic methods, the solution is obtained fairly quickly, but the results are only as good as the rules used.

Mignon et al. (1995) describe a framework where schedules from an optimal scheduling package are automatically passed to a Monte-Carlo based simulation implemented within the BATCHES simulator (Clark and Joglekar, 1992). Performance and robustness criteria are defined and used to compare strategies such as following the original scheduled timings, allowing forward shifting of start times where feasible, and advancing the due dates in the scheduling model in an attempt to create a more robust schedule. It was found that advancing the due dates improved both the expected performance and the robustness of the schedule.

Honkomp et al. (1997) extend this framework so that a deterministic simulation model runs in parallel with the uncertain model. The two may then be compared as a basis for initialising rescheduling, although this is limited at present to delaying the start time for infeasible tasks. Discrete time and continuous time scheduling algorithms were compared on a number of problems, with strategies such as using the nominal processing times, extending the duration of bottleneck tasks, and extending the duration of the final tasks in each recipe. The results show that the best algorithm and strategy varies with the problem, but that it is often an improvement to increase processing times from the nominal ones under uncertainty, although the busiest point in the schedule may not be the best point to do it.

2.3 Supervisory Control

Supervisory control is that function which fills the gap between production scheduling and low level sequential control. Rippin (1983) describes how even when the sequence of tasks in a plant is fixed, rules for resolving conflicting demands for the same equipment item or other resource may be required.

Various representations of the plant and process have been proposed. Arzen (1994) presents an approach based on the Grafcet representation. This is a sequential function chart type representation and is implemented within an expert system environment. Gonnet and Chiotti (1997) present a model for a supervisory control system based on petri nets designed for use in situations where the plant topography changes during operation. However the development of international standards in batch control, which will be described in Section 2.5, have gone a long way in standardising the format of the models used.

The core functionality of a supervisory system is therefore to initiate commands in a
control system in the correct sequence so that a given schedule is carried out. An example of a system which performs purely this key function is presented by Johnsson and Arzen (1998). Such a system ensures that sequences of control actions are only initiated in the control system when the required resources are available.

However, such a system is running effectively blindly, with no knowledge of when events are likely to take place and no look-ahead capability. Cott (1989) describes a system for managing the execution of a schedule in a plant control system that keeps a real time Gantt chart updated with the proposed actions. Keeping this chart up to date in response to disturbances, such as variations in processing times, is known as on-line scheduling, which will be described further in the next section.

The original system has since been developed into the SUPERBATCH software (SUPERBATCH User Manual, 1996) (Macchietto, 1996). SUPERBATCH reads a set of input files in the format described in Appendix B. These describe the plant and the recipes, together with the batches to be executed. It then calculates detailed timings for each step and displays the result as a Gantt chart. The on-line version then communicates with a plant control system initiating the execution of steps within that system when it is feasible for them to begin and receiving notification when they are completed. The schedule is recalculated each minute so that deviations from the nominal processing times given in the model that occur when steps are actually executed are automatically accounted for.

A key feature of any supervisory control system is the fact that it is a real time system that must be able to cope with industrial scale problems, and therefore the algorithms used must be extremely rapid (Gonnet and Chiotti, 1999). There are a wide range of disturbances that may occur in a batch plant, from minor variations in the processing times of individual steps and small deviations from the amounts prescribed in the recipe, to equipment failure, changes in resource availability, (such as a operator not reporting for work), and changes in the order slate.

Dealing with such disturbances as and when they occur, (rather than trying to predict and plan around them, which was dealt with in Section 2.2.4), is often dealt with as a separate function in itself, which can then be embedded within a supervisory system.

2.3.1 Rescheduling and On-line Scheduling

At one end of the spectrum there is work on near instantaneous routines which react to disturbances detected in the process.

The P.O.M.A. (Projected Operation Modification Algorithm) and T.F.M.A. (Transfer Fraction Modification Algorithm) routines (Cott, 1989) were some of the first described and are at the heart of SUPERBATCH. The P.O.M.A. adjusts the start times of each step in response to deviations from expected plant performance using the same earliest
completion time algorithm that was used to calculate the initial schedule, but taking into account batches and operations which have already taken place. The T.F.M.A. adjusts the amounts used in future steps in response to changes in the actual amounts used so far. These algorithms can be run on a minute-by-minute basis to ensure that the production plan remains up to date and feasible.

The routines within SUPERBATCH do not consider altering the equipment assignments, and Ko and Moon (1997) present a number of algorithms for rescheduling extending those described by Cott. Firstly the time shifting can be reversed so as to minimise the storage of final product. Secondly parallel equipment is considered. However it is assumed that there is parallel idle equipment, which is unlikely to be the case.

As soon as the rescheduling becomes more complicated than simply shifting timings or quantities, the algorithms become more complex, and move away from being a real-time tool, while remaining very fast. Kanakamedala et al. (1994) describes a least impact beam search to reschedule in the event of conflicts or delays during the execution of a schedule. A decision tree of all possible reroutings of the product in conflict is created, and this is then pruned using heuristics such that the overall disruption on the rest of the schedule is minimised. Huercio et al. (1995) also proposed modifications to the batch sequence, as well as changes in equipment assignment, in a reactive scheduling tool. This is also based on a heuristic decision tree analysis.

Kim and Lee (1997) present a two stage system, the first part of which monitors the execution of a schedule for a deviation in the starting or finishing time of a batch from the original plan. In this event the effect on the other batches in the schedule is calculated and if a coefficient based on the shift in the timings of the other batches is exceeded then the second stage in the system is executed. This is a rule based system for shifting start times, re-allocating equipment or re-sequencing the batches, and performs the user-given rules in an attempt to correct the deviation from planned production. The on-line scheduling is therefore triggered by deviations rather than being periodic as in Cott’s system.

Where there are both continuous and batch units, the problem of setting parameters for both becomes very involved. Djavdan (1992) uses a dynamic simulation model to determine the throughput of continuous units and the starting time of the sequences of batch units.

At the other end of rescheduling from on-line scheduling, optimisation techniques may be used to re-optimise the schedule. Chua (1995) describes extensions to an MILP scheduling formulation to allow for partially executed tasks, and to alter the objective function in order to minimize the degree of change from an original solution. This may be important where there are many manual processes, since operators are likely to be unhappy with a constantly changing set of tasks in the near future. However as the degree of automa-
tion within a plant increases, minimising the changes in the schedule is likely to be less important.

Rodrigues et al. (1996) use an STN representation but a reduced MILP formulation for rescheduling that does not consider material balances or equipment assignments, and assumes the number of batches is fixed. A number of small parts of the horizon are considered in turn so that the overall solution time is small. However the method is not suitable for problems involving finite intermediate storage due to the absence of material balances, and if the equipment allocation is fixed only the timings of tasks can be shifted.

A balance between fully resolving the scheduling problem against resolving parts of the problem is provided by Schilling (1997), who presents a multi-level approach to rescheduling. This is based on using a previous solution as far as possible in order to reduce the computational effort required. At the first level the integer assignments are kept the same and therefore only one LP needs to be solved, which is usually very rapid. Gradually various aspects of the original solution, such as the timings of each task, are relaxed until the problem returns to recalculate the full schedule. This process may easily be carried out in parallel, since each level is independent, which would speed the solution in cases where the initial level does not result in a feasible solution.

The introduction of a rescheduling system in an industrial setting may require care, as mentioned by Whitworth (1993). This paper describes a methodology for first introducing a rescheduling system off-line, while using real plant data, in order to evaluate its performance before introducing it on-line. Initially the rescheduling system is run against a simulator, which is run in parallel to the real process. The results of this build confidence in the rescheduling system and iron out any difficulties before it is introduced as a fully closed loop system acting on the real process.

2.4 Integrated Operations

So far the individual parts of the hierarchy have been examined. However these separate applications must work together (Bassett et al. (1996), Puigjaner and Espuña (1998)), if an organisation is to avoid so called “islands of automation”, and achieve something close to optimal overall operation.

One of the first cases of formal methods being developed for passing information from scheduling to supervisory control is presented by Crooks (1992). The initial aim of this work was to automatically develop detailed sequences for plant operation, and the CAPS. (Computer Aided Procedure Synthesis), system was developed. However by using the same STN model used by CAPS to generate master procedures and control sequences for scheduling in gBSS, then the optimal schedules can be automatically translated into sequences of master procedures, with unit allocations, as SUPERBATCH input.
Chua (1995) then looked at gBSS and Superbatch models with a view to integrating them by defining the mapping between them, therefore applying the data flow approach defined by Stanley (1994). He then went on to extend the MILP model of Kondili et al. (1993) to allow partially completed tasks to be modelled when rescheduling.

Both the approach of Crooks and that of Chua require fairly detailed gBSS models. This leads to one of the major problems cited with an integrated system which is the solution time of the MILP scheduling package. On the one hand the continual improvements in the algorithms used, advances in solvers and processor speeds, as well as developments such as parallel computing (Subrahmanyam et al., 1996), mean that the size of scheduling problems which may be solved in a reasonable length of time is increasing. However the exponential increase in solution time with problem size means that any way of keeping the problem size small will greatly improve performance, and there is little room for the model being any larger than absolutely necessary.

![Diagram of Architecture of an advanced operations support system (Liu, 1995).](image)

Liu (1995) describes an advanced integrated manufacturing system for a batch pilot plant. The systems developed are fully integrated so that supervisory control, sequential and non-linear controllers, and a dynamic plant model, are all used together to improve plant performance. However the link between off-line scheduling and the rest of the system remains manual. The overall system developed is shown in Figure 2.5. The system consists of two branches, one the real plant, the other a simulation, either of which may communicate with a central core of functions. The simulation can therefore
be used to test actions, before they are carried out in the real plant.

Ishii and Muraki (1997) present a framework for on-line scheduling and supervisory control in a very similar manner to Cott (1989), and also describe a framework where the controller is linked to an off-line simulation.

Elsewhere, the work of Mignon et al. (1995) and Honkomp et al. (1997) in examining the effects of uncertainty on schedules are some of the few examples of the translation of optimal schedules for use in another package being described in the wider literature. However this work focuses more on the analysis of the schedule than on a framework where that schedule may be implemented within an actual control system. It also only describes the one way flow of information from scheduling to a simulation without any return loop for re-initialising the scheduling model.

Papers such as Rehbein and Andrews (1996) show that the set of applications applied to batch operations are not always identical to the hierarchy described. In their paper a Distributed Control System (DCS) is interfaced directly to an Enterprise Resource Planning (ERP) system. ERP systems tend to be mainly transactional based systems with functionality in a wide range of business areas from finance to human resources. In this case there is a scheduling module which is interfaced so that control recipes may be down-loaded straight into a DCS. Such a system skips any supervisory management of the schedule and may perform badly where there are large disturbances. This paper also highlights that as well as theoretical challenges in integrating applications there are often large practical difficulties associated with enabling different hardware platforms and operating system to exchange information.

An example of a completely integrated decision support system reported in the literature is the YFADI system (Adamopoulos et al., 1994) designed for a textile plant. This ties various scheduling, planning, execution, and procurement functions into a single database allowing all applications access to the information (but not models) that they require, and is an example of the information management mode of integration. A more general system is described by Puigjaner and Espuña (1998), who present a fully integrated system of applications, also based on the hierarchy shown in Figure 1.1. Each application is tied into the single database so that their knowledge based algorithms have instant access to the latest information.

In an overview of the general requirements for CIM applications, Kappel and Vieweg (1994) define the following data types; Product, Production, Operational, Resource, Financial and Marketing. They also note that the requirements of the database are that it can be manipulated, it is independent from the applications, it can recover from systems failure, and that certain authorisations are required for access and manipulation.

Although not directly concerned with the integration of scheduling and supervisory
control, several authors present systems that are of interest from a modelling point of view. For example, Fritz and Engell (1997) describe a single modelling environment in which batch simulation problems can be described. A range of solvers can then be called depending on what problem needs to be solved. This shows how useful it can be for a user to be presented with a single modelling environment, and the formatting of the problem for a particular solver to be left to automatic routines.

Simensen et al. (1997) present a multiple view information model for batch plants. This allows various users to view the information they require for a particular action by extracting it from a single “meta” model. These views include recipe management, production planning and scheduling, process management, process control and engineering. As stated by Mannarino et al. (1997), it is often the lack of comprehensive models in an enterprise that leads to a lack of integration and islands of automation.

2.5 International Standards

Various attempts at standardisation in batch processing have been attempted. The S88.01 standard (ISA, 1995) from the Instrument Society of America defines reference models for batch control together with terminology that explains the relationship between these models and terms. The aim is a common language for describing batch processing.

There are four main parts to the S88.01 model, the process model, the physical model, the procedural model, and the recipe model, each of which is a hierarchical structure as shown in Figure 2.6.

The process model describes the subdivisions of a batch process. The process consists of a set of process stages, describing a part of the process that usually operates independently from other process stages. For example the production of PVC from the Vinyl Chloride monomer might have the stages “Polymerize”, “Recover monomer” and “Dry”. The process stages consist of sets of process operations, each representing a major processing activity. In the PVC process the Polymerize process stage might consist of operations such as “Prepare reactor”, “Charge” and “React”. Each process operation is divided into process actions, each representing a minor processing activity. Example of process actions in the process operation React might be “Add”, “Heat” and “Hold”.

Figure 2.6: The S88.01 Standard model structure.
The physical model starts with an enterprise, which may contain sites, which may contain areas. However the boundaries of these entities are outside the scope of the S88.01 standard. The standard does set boundaries for the process cells which may be contained in an area. A process cell contains all of the units, equipment modules, and control modules required to make one or more batches. A unit contains all the necessary physical processing and control equipment required to perform one or more major processing activities. An equipment module can carry out a finite number of specific minor processing activities such as dosing and weighing. A control module is typically a collection of sensors, actuators, other control modules, and associated processing equipment that, from the point of view of control, is operated as a single entity.

The procedural control model is headed by a procedure, which defines the strategy for carrying out a major processing action such as making a batch. It is made up of unit procedures, which are ordered sets of operations that cause a contiguous production sequence to take place within a unit. An operation is an ordered set of phases that defines a major processing sequence that takes the material being processed from one state to another.

The recipe model starts with a general recipe, which specifies material requirements but is created without specific knowledge of the process cell equipment that will be used to manufacture the product. The site recipe is the combination of site-specific information and a general recipe. The master recipe is that level of recipe that is targeted to a process cell or a subset of the process cell equipment. Recipes contain the following information: header, formula, equipment requirements, procedure, and other information.

The management of these different recipes and the process of converting one to another has led to the development of specialised software, e.g. that of Wang et al. (1995) and Wang et al. (1997), which helps a user with the task of creating one sort of recipe from another.

The follow up to the S88.01 standard is the S88.02 standard, currently in draft form (ISA, 1997). This attempts to define data structures and guidelines for languages for use in batch control based on the models defined in S88.01. The aim is the creation of Batch Interchange Tables, (BITS), which will be in the form of a Structured Query Language, (SQL), database, that can be read and/or written to by all the different batch control software packages.

Another draft standard from the ISA is the S95.01 ISA (1999), which looks at models for the interface between computer integrated manufacturing and wider business functions. The standard at present has a much wider scope than batch control but is therefore much more vague. At present it seems to draw a line between enterprise-wide planning and short-term scheduling, but does not really define what should cross that line.
2.6 Discussion

This chapter has shown that there have been large research efforts in the areas of planning, scheduling and supervisory control, and there are now many commercial applications in these areas (e.g. Pinto et al. (1998), Shah et al. (1995)).

The development of scheduling technology has led to the generation of both general and specific algorithms which can solve ever larger problems within a reasonable time. The concurrent developments in computer technology allow further reductions in solution times. Therefore many of these methods have become practical for industrial implementation.

The work on including uncertainty into the initial scheduling problem is less developed. There are also issues about how reliable any uncertain parameters may be. Although the incorporation of such information into the scheduling formulation may lead to schedules which require less modification if something does go wrong, it is likely that rescheduling will be required in any case, and if nothing goes wrong the schedule may be conservative compared with one arrived at ignoring the uncertainty. Deviations that do not require major rescheduling can be simply adjusted at the supervisory control level.

Much less work has been published in the area of applications integration. There is therefore scope for producing an integrated system for scheduling and supervisory control, based on a single data model. The key advantages of such a model would be as follows.

- The single model ensures consistency. If the models for the various applications come from a single source, there is no chance that one might be out of date or incomplete.

- Less repetitive effort to create the system. The considerable overlap of information between the various applications is eliminated.

- Easy to maintain/change the system. Often the recipe management and set-up time are a very significant factor in profitability (Lemetais, 1999), and therefore a system where information and models are stored in a single place offers advantages over a number of distributed models. In this approach the addition of a recipe requires a single model to be updated. Other approaches require all sub-applications as well as mapping routines, or a central database, to be modified.

- Single approach company-wide. Model based algorithms can be applied to any plant, and therefore all of a companies plants can be predicted to be operated in a similar manner. This will also aid in the integration with company wide systems.

The creation of a complete system for supporting plant operation represents a departure from traditional areas. Firstly off-line scheduling must include details of current plant
activities, while few if any works published on scheduling methods show any work that does not assume the plant is static at the beginning of the horizon.

Secondly, rescheduling rarely involves any change in the order slate, but merely applies to adjusting the timing, and/or the resource allocation of the current schedule. In the real world scheduling is often not as well defined as standard optimisation problems, and it is unlikely that many occasions will occur where all production ceases, which is the usual assumption in papers on scheduling. Throughout this thesis, scheduling and rescheduling will be used to mean the generation of a sequence of batches to meet some aims, with the current plant status as an initial position. In effect all scheduling is rescheduling, only the degree of new information may vary, from some disruptions in processing times to a large influx of new orders. The adjustment of the current schedule in real time is termed “on-line scheduling”.

2.6.1 Information Requirements

At the supervisory control level the development of the S88.01 standard and its widespread adoption present this model as a basis which must be adhered to when considering a single model for batch operations.

For short-term scheduling, the STN representation is almost a de facto standard, at least among mathematical programming based methods, given its widespread and continued use in the literature. Should the RTN representation become more widespread, then the similarities with the STN should mean that this will not pose any difficulties.

The next chapter therefore presents a single data model that will form a basis for an integrated decision support system.
This chapter describes the structure of a model that will be used as a basis for integrating off-line optimal scheduling and on-line supervisory control. The model is therefore based on the models used for off-line optimal scheduling and on-line supervisory control. For the purposes of this work, the scheduling package will in the main be gBSS and the supervisory control package will be SUPERBATCH. The aim is therefore for the model to be able to capture all of the features that may be modelled in these generic packages, with a minimum of repetition. (The models and input languages for gBSS and SUPERBATCH are described in Appendices A and B respectively).

In fact the range of plants for which any resulting integrated system is applicable will be limited by those features which may be modelled. Table 3.1 describes a wide range of the common features of batch plants which should be handled, and represents the target for what the common data model should be able to describe.

In addition to the fundamental structure of the model, a definition language for this model is also described. This not only provides a vehicle for entering such a model into a database, but also shows in detail where the various parameters are kept in the model.

3.1 Model Overview

The model has three main parts: equipment, materials and procedures. This closely follows the format of both the gBSS and the SUPERBATCH models, as well as the S88.01 standard. As far as possible the definitions of equipment and procedures are kept separate, so that changes in the equipment available do not affect the definition of procedures, and vice versa. The three areas are now discussed in turn.

3.2 Equipment

Equipment is treated in a two level hierarchy as shown in Figure 3.1, reflecting the structure used by SUPERBATCH. The basic element is the unit, which is generally identifiable
Chapter 3. Model Structure and Definition

<table>
<thead>
<tr>
<th>Feature</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of units</td>
<td>One</td>
</tr>
<tr>
<td>Number of products</td>
<td>Many</td>
</tr>
<tr>
<td>Product structure</td>
<td>Independent</td>
</tr>
<tr>
<td>Intermediate stability</td>
<td>Stable</td>
</tr>
<tr>
<td>Intermediate storage</td>
<td>None (NIS)</td>
</tr>
<tr>
<td>Cleaning/product changeovers</td>
<td></td>
</tr>
<tr>
<td>Equipment structure</td>
<td>Single-path</td>
</tr>
<tr>
<td>Equipment connectivity</td>
<td>Full</td>
</tr>
<tr>
<td>Equipment usage</td>
<td>Single use</td>
</tr>
<tr>
<td>Shared routing</td>
<td></td>
</tr>
<tr>
<td>Processing times</td>
<td>Fixed</td>
</tr>
<tr>
<td>Mode of execution</td>
<td>Batch</td>
</tr>
<tr>
<td>Transfers</td>
<td>Sequential</td>
</tr>
<tr>
<td>Constraints on task sizes</td>
<td></td>
</tr>
<tr>
<td>Common Resources</td>
<td>Integer</td>
</tr>
<tr>
<td>Recycle of material</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Limited (LIS)</td>
</tr>
<tr>
<td></td>
<td>Unlimited (UIS)</td>
</tr>
<tr>
<td></td>
<td>Shared (SIS)</td>
</tr>
<tr>
<td></td>
<td>Linear with batch size</td>
</tr>
<tr>
<td></td>
<td>Non-linear with batch size</td>
</tr>
<tr>
<td></td>
<td>Semi-continuous</td>
</tr>
<tr>
<td></td>
<td>Continuous</td>
</tr>
</tbody>
</table>

Table 3.1: Features of Batch Processes.

as a separate processing item in which one or more major processing activities may be conducted, and to this extent is identical to the definition in the S88.01 standard (ISA, 1995).

However other items that are critical in scheduling the process, such as routings, may also be modelled as units, whereas in S88.01 they would strictly be defined as “equipment modules” or “control modules”. An example of this is shown in Figure 3.2. The valve shown is critical in deciding whether a transfer can take place between say tank A and tank C. If a lengthy transfer is taking place between tank B and tank D, a transfer between tank A and tank C may be considerably delayed, and as such the valve may play a critical part in how the processes are scheduled. A possible solution if S88.01 ‘purity’ is to be

![Figure 3.1: Structure of equipment items in the unified model.](image-url)
maintained is to use the words "equipment module", "control module" and "unit" when dealing with a user, e.g. in input files, but to treat all these items identically internally.

![Diagram of route critical valve](image1)

**Figure 3.2: Example of a route critical valve.**

A resource is a collection of one or more units that are functionally identical. Recipes are defined in terms of these resources, and this simplifies the description where there are a number of identical units. For example if there are three identical mixers, it is easier to say that a mixing operation is performed in a "Mixers" resource containing the three mixers, rather than to explicitly name each mixer. When batches are scheduled in SUPERBATCH, the specific units in each resource must be specified, i.e. at this stage the actual mixer must be specified from the collection of three. The model structure where each unit is a member of only one resource means that if the schedule created by gBSS uses a unit in a recipe, then that usage automatically specifies the unit/resource pairing for that recipe.

Figure 3.3 shows a flowsheet of a simple example plant, consisting of two raw material tanks, a mixer, a reactor and a product tank. Table 3.2 shows how this information is specified as resources and units for this example.

Each resource has a name, a type and higher level specification. The type is one of the following:

- **FEED** - A resource that takes material inputs from outside the site, where the site is the limit of what is being modelled. (This S88.01 standard defines the terminology

![Diagram of example flowsheet](image2)

**Figure 3.3: Example flowsheet.**
Table 3.2: Definition language for equipment

of “sites” and “areas” as physical, geographical or logical groupings, but does not define the criteria for defining their boundaries. Site is chosen as the best word for what is being modelled.) An example of a feed resource would be a raw material storage tank, since this receives inputs when raw material receipts arrive at the site.

- **PROCESS** - This is the standard type of batch processing unit, where batch integrity is maintained.

- **STORAGE** - These units allow the mixing and splitting of material between a number of batches. Processing steps may also be carried out in a storage resource.

- **PRODUCT** - A resource that gives material outputs away from the site.

These types are the same as those used in SUPERBATCH. The next tag, **HIGHER LEVEL**, for each resource describes the way in which the units in that resource are to be modelled at the higher, gBSS, level. The range of alternatives is as follows:

- **UNIT** - These units will appear as units in both the SUPERBATCH and the gBSS models. This is the normal tag for most units.

- **IGNORE** - These units will not appear in the gBSS model, although they are included in the SUPERBATCH model. This might be used for units which are only used for short periods and therefore cannot be efficiently modelled in gBSS with a reasonable time discretisation interval. Additional constraints, such as shared routing, may be removed from the gBSS model simply in order to reduce the problem size, although the problem being solved becomes more of an approximation of the real situation.
Chapter 3. Model Structure and Definition

- **COMMON RESOURCE** - This tag is for units which are modelled in SUPERBATCH as being on a route during a transfer or other operation, and must therefore appear in gBSS as a common resource during that period, since gBSS assumes only one unit performs a task. This distinction would not be required if the Resource Task Network formulation (Pantelides, 1994) was used, since the treatment of units and common resources is then identical, both being regarded equally as resources.

The way a resource is tagged is not fixed, and a user may experiment with different representations while setting up the model. For example a scheduling model may initially be found to be too slow with all the routings modelled, so the gBSS time discretisation size might then be increased and the routing valves then switched to IGNORE since it no longer makes sense to include them in the model. Since only the gBSS model is affected by the HIGHER LEVEL tag, these tags could even be changed during operation of an integrated system if necessary, so long as a completely new gBSS model is created before executing the scheduling package.

Each unit has a name and a size. No units are given for the size, and it is simply left to the user to ensure that the units for equipment and recipe sizes are consistent across the model. However this highlights another advantage of a single model approach to integration, since it is only across one model that the units have to be consistent, rather than across the models used by each application. Future versions of the system could be extended to support data units management.

The material(s) which may be stored within a unit must be specified, as shown in the unit Store_tank_1. This information is required by gBSS so that storage constraints are satisfied, whereas SUPERBATCH moves material between vessels as directed, and only checks that vessel capacities are not exceeded. The material stored is specified as a “state”, as defined in section 3.3.

3.2.1 Continuous Equipment

Many processes are not entirely operated as either batch or continuous but contain units operated in both these modes. For example a packing line may operate continuously, whereas the process manufacturing the material being packed operates in a batch mode, resulting in a so called semi-continuous process.

In order to correctly model and handle continuous units, a resource may be labelled as being of a CONTINUOUS type as described in Table 3.3. The unit is specified as having a RATE, (in units/hour), rather than a SIZE and may have an additional MIN_RUN_LENGTH, (in hours), specified as the minimum length of time that it must run without stopping, since the repeated starting and stopping of a unit may be undesirable.
3.2.2 Common Resources

Common resources are those items in a plant which may be used by a number of different processes, for example steam supplies, operators etc., and are defined as shown in Table 3.4.

This is similar to the definition in gBSS or SUPERBATCH, but adds the optional SUPERVISE ONLY tag, which removes the resource from the gBSS model. This allows for the possible removal of the constraint in cases where it is not thought necessary to include it in optimal scheduling, but the constraint nevertheless exists and must be satisfied during on-line execution.

3.2.3 Equipment Flowsheet

SUPERBATCH checks for connectivity between units and the flowsheet definition required is repeated in the combined model, with the same language structure, as shown in Table 3.5.

3.3 Materials

Materials are modelled in a three layer hierarchy, as shown in Figure 3.4. This represents a combination of the way in which materials are represented in SUPERBATCH and gBSS.

SUPERBATCH represents materials as generic material types, which are made up of one or more specific materials. This allows the definition of a single recipe to represent a
number or products which only differ slightly in their ingredients, e.g. different flavours in a yogurt plant. The generic material would be fruit, with perhaps banana, cherry and strawberry as specific materials. Recipes will be defined in terms of generic materials, and then the specific material for each generic material in the recipe must be specified when the recipe is scheduled for execution.

gBSS represents materials as “states” in the State Task Network process description. Each state may differ not only in the chemical composition of the material but also in its location or physical attributes. The state therefore represents an extension of the specific material, since each specific material may exist as a number of states, e.g. a specific material, for example cherry_paste, may have hot and cold states, the cold_cherry_paste needing a heating task to become hot_cherry_paste.

An example definition is shown in Table 3.6. States may be tagged as STABLE, as in gBSS, denoting they may be stored in the processing vessel that produced them. Taking the previous example, cold_cherry_paste may be stable, while hot_cherry_paste is unstable since it will cool down if kept waiting.

<table>
<thead>
<tr>
<th>GENERIC MATERIAL additive</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIFIC MATERIAL cherry_additive</td>
</tr>
<tr>
<td>STATE bulk_cherry</td>
</tr>
<tr>
<td>STATE cherry_weighed STABLE</td>
</tr>
<tr>
<td>SPECIFIC MATERIAL strawberry_additive</td>
</tr>
<tr>
<td>STATE bulk_strawberry</td>
</tr>
<tr>
<td>STATE strawberry_weighed STABLE</td>
</tr>
</tbody>
</table>

Table 3.6: Definition language for materials

This completes the material specification.
3.4 Recipes and Procedures

The description of procedures starts with the definition of the operations which may be performed in the plant, as required by both SUPERBATCH and S88.01 standard.

An operation is simply a processing activity which may be performed in the plant, and is an independent entity, not part of any recipe. These are defined in the same way as in SUPERBATCH, as shown in Table 3.7.

```
OPERATION charge_r
   TRANSFER FROM RESOURCE raw_store THROUGH feed_pump TO reactors
   USE COMMON RESOURCE electricity
   ENC OP

OPERATION react
   PROCESS IN RESOURCE reactors
   ENC OP
```

Table 3.7: Definition language for operations

Once the operations are defined these will be pieced together to form the master recipes. Each master recipe is made up of one or more tasks, which contain one or more steps, as shown in Figure 3.5. Each step is an instantiation of an operation.

The header information for each master recipe is defined as shown in Table 3.8, following the requirements of SUPERBATCH and S88.01. The recipe itself is then defined as a list of steps, each of which may appear in one or more tasks.

As an example of a recipe definition, the following recipe, which will be performed in the example plant shown in Figure 3.3, is modelled. Material A is charged to a mixer, where it is then heated. Material B is then charged to the mixer, after which the two materials are mixed. The mixture is then transferred from the mixer to the reactor. There

![Figure 3.5: Structure of recipes in the unified model.](image-url)
it is heated to a set temperature, after which it is held at that temperature. Finally the product is discharged from the reactor to a storage tank.

This recipe is shown in a graphical form in Figure 3.6, as a Sequential Function Chart, (SFC). This shows which operations are performed in which resources over the execution of the recipe. The grouping of steps into tasks is also shown. The recipe consists of two tasks and eight steps. (The STN for the process is shown later in Figure 3.10).

The task MIX contains five steps, while the task REACT contains four steps. The operation TRANS_AB appears in both tasks in order to model the transfer in gBSS, as described by Liu (1995). The gBSS model will then represent the two tasks as shown in Figure 3.7.

At time A the transfer would start from the mixer to the reactor. However the state MIXTURE is only released by the task MIX at time B, when there is a corresponding requirement for this state into the REACT task. Times A to B therefore represent the transfer duration. Should this duration be much smaller than the time discretisation in the gBSS model, then this gap will disappear when the task durations are rounded to the nearest interval, and the transfer will be modelled as being instantaneous.
Chapter 3. Model Structure and Definition

Table 3.8: Definition language for master recipes

```
MASTER RECIPE make_prod
VERSION 1.0
DATE 73/07/23
AUTHOR Alexander.the.Great
BATCH SIZE NOMINAL 5000.0
MIN 1000.0
MAX 5100.0

MATERIAL bulk
MATERIAL additive

RESOURCE raw_store
RESOURCE reactors

COMMON RESOURCE steam

BEGIN
    list of steps ...

END
```

Figure 3.7: Modelling transfers in a discrete time representation.
This way of modelling the transfer is consistent with stable states, since the material can for example remain in the mixer for a period before being transferred to the reactor, as shown in Figure 3.8, with the storage of the stable intermediate shown as a dotted line. However the usage of common resources during the transfer should only appear in one task, and to be entirely correct they should appear at the beginning of the REACT task, since this will be when the transfer is actually taking place.

The recipe is defined in terms of the resources used, rather than units. The exact units to be used are only specified when the recipe is to be executed, and this information will be taken from the units used in the schedule generated by gBSS.

A task is therefore a linear collection of sequential steps. However this is unlikely to be a limitation in the vast majority of cases. Any branching in the sequence of steps that involves more than one resource will require more than one task, since the STN representation demands that a task be performed in a single unit. Where a unit must perform two steps simultaneously, the STN will not allow two tasks to be performed simultaneously, and even the RTN formulation would be very contrived. However this could be overcome if the shorter step, or sequence of steps, is ignored by all tasks, and simply appears as an isolated task in the recipe.

An example might be where a tank is cooled, and during the cooling step a step instructing a sample to be taken by an operator is to be executed. The sampling step may appear in the recipe unattached to any task, and therefore be ignored in the gBSS model, while remaining in the SUPERBATCH model. However in the vast majority of situations the representation of tasks as linear collections of steps would seem straightforward and practical.

The language definition for the steps in our data model is shown in Table 3.9. The basic definition follows that of SUPERBATCH. However, each step is also described as

---

1 A task could produce a dummy unit initially, which is consumed initially and uniquely by the second task, but even this has pitfalls.
being in one or more tasks, which corresponds to the task boxes the steps are within in Figure 3.6.

The step contains a list of the resources that are used. Where this resource is the resource that carries out a task containing this step, the relevant input or output states are defined. For example, STEP_1 is in task MIX which is a task performed by the units of the resource MIXERS. Therefore the resource description for MIXERS in this step includes the line IN-STATE A, since state A enters the task during this step.

Where the resource is not the resource that performs the task, as in the case of Storage_A in STEP_1, the line INITIAL-STATE A denotes that the state A is leaving this resource, but that it is not an input state to a task being performed in this resource.

The other type of statement which may appear in the resource description is the PRECONDITION statement. This is used to limit the conditions under which a unit may perform a task. When a unit performs a task it is assumed to switch to a condition equal to the last state to leave that task. In this case when a mixer performs MIX it switches to the condition MIXTURE, and when the reactor performs REACT it switches to the condition PRODUCT. A cleaning task might leave a unit in condition WASTE_DETERGENT. Therefore if this product can only be made after the mixer is cleaned or the same product is made, then the preconditions are MIXTURE and WASTE_DETERGENT. If cleaning must always take place before a batch of this product, then WASTE_DETERGENT should be the only precondition. If no precondition lines appear, the task is assumed to be performed under any condition.

The duration and transfer quantities in the step definition may be tagged as being either a FIXED AMOUNT or a BATCH SIZE FACTOR, depending on whether the quantity is fixed for any batch size or else varies linearly as batch size varies from the nominal size. This is identical to SUPERBATCH.

Where there are simultaneous transfers, the order of input states and output states in the resource description should mirror that of the lines of transfer amounts, so that the proportions for each input state and output state can be correctly assigned.

The recipe definitions conclude the definition of the global data model.

3.4.1 Limited Connectivity

Where there is limited connectivity between units, the State Task Network must be extended to take account of this, since gBSS assumes that there is complete connectivity between units. For example, the plant shown in Figure 3.9 is an extension of the plant shown in Figure 3.3 and requires that the STN be extended as shown in Figure 3.10. In this case material produced by tasks in Mixer.#1 cannot be consumed by a task occurring in Reactor.#2.
Table 3.9: Definition language for steps in master recipes
Figure 3.9: Example flowsheet of a plant with limited connectivity.

The input language has extensions to allow this to be modelled, as shown in Table 3.10, which shows the changes in \textit{STEP} 3, transferring material from a mixer to a reactor.

For this case the same single recipe is used, but it now contains four tasks, although the steps remain more or less the same. The tasks are dedicated to a unit, (they could also be dedicated to a number of units if necessary), and the input and output states are specifically linked to a task. Together with adding the extra units to the equipment definition and changing the flowsheet, these are the only changes that are required to the model to take account of the extra units. Had there been complete connectivity between the mixers and reactors, only the equipment definition would have had to be changed.

\textsc{Superbatch} automatically checks for connectivity between units, only allowing transfers through feasible paths, and therefore only requires that the plant flowsheet correctly describes the situation.

The limited connectivity could have been found automatically from the flowsheet, and the STN then automatically extended. This would simplify the model definition where limited connectivity exists. However designing and implementing such algorithms would be a lengthy process, and does not aid the validation of this method of integration.

Figure 3.10: State Task Networks for example plants with and without limited connectivity.
Table 3.10: Definition language for steps with limited connectivity

3.5 Additional Information

Additional information that will be used in the construction of the rest of the model and the operation of the integrated system is specified as shown in Table 3.11. This information consists mainly of parameters that will be used generally, and is therefore not attached to any of the previously mentioned data objects.

Firstly the site name identifies this collection of equipment from any others in the enterprise. The situation name allows different situations for the same site to be differentiated. This name is used whenever possible in the output of the various applications.

The rounding coefficient is used whenever timings have to be rounded onto the discrete times in gBSS. The coefficient given is between 0 and 1, where 0 always rounds down, 0.5 is standard rounding to the nearest whole number, and 1 always rounds up.

The variability coefficient is used in a simulation model that will be generated. The variability in each processing time is a normal distribution around the nominal time with a
standard deviation equal to this fraction of the processing time. This is described further in Section 4.4.

The inviolate period is a conservative estimate of how long off-line optimal rescheduling is likely to take. This is then used as a lookahead period, since the actual production is likely to have advanced by this amount by the time off-line scheduling is complete.

The minimum and maximum recipe sizes given in the recipe header can be used to constrain the batch sizes in units in the gBSS model. This option is turned on by the “USE MIN AND MAX UTILISATIONS” flag. Paradoxically adding these constraints was sometimes found to increase gBSS solution times, while in other cases it reduced them. Should a physical constraint exist, for example a stirrer must be covered, or recipes are only ever executed at a single size, then this flag should be applied to ensure that gBSS always returns feasible schedules. Also by ensuring that the minimum utilisations are greater than half the equipment capacity, then batch splitting is effectively removed as a potential for error. For example if the plant in 3.9 had full connectivity, gBSS might return a solution where material from a mixer is split and processed in both reactors, but this is not possible if the reactors must be more than half full.

Further information includes the parameters and options for any of the scheduling packages that may be used.

3.5.1 Initial inventory and order slate

Initial state information is described as shown in Table 3.12. A state may have an initial amount in a specified tank, and a price which will be used in the gBSS model, though neither of these is required. Delivery orders follow the gBSS style and are either for an exact amount or for a given range. The time may be specified precisely in which case a value is given for each unit of material successfully delivered, or may be required over some range of time in which case a penalty value for delivery after the initial time is given together with a penalty form of either LINEAR or SQUARE depending on how that penalty is to increase.

Additionally orders may be specified as being PLACED AFTER a given time, in which case they will not be included in any scheduling model until after the specified period. This allows a scenario of additional orders to be set up initially and the new orders to be added without any manual intervention.

Timings are either given as a date format of HH:MM DD/MM/YYYY or else as an offset from the current time, so that input files do not continuously become out of date during testing.
3.6 Scope and range of the model

This model therefore allows any combination of the features outlined in Table 3.2, and preserves just about all the modelling flexibility available in gBSS and SUPERBATCH. Those parts of the gBSS and SUPERBATCH models not included are specific to that model, and not an issue in integration. These could be easily added to the model as additional attributes to the existing objects. For example, gBSS attributes various costs, such as those for using a common resource, and SUPERBATCH lists parameters for recipes. These features could easily be added to objects in this data model, but leaving them out speeds the development of the prototype system.

Overall the model structure is fairly close to the SUPERBATCH model, with additional material specifications, and another layer in the recipe structure. The full structure is shown in Figure 3.11, with items that will be added later during the operation of the system shown in red for future reference. A single manager class is used to hold the sets of each type of object.

Now that a model structure has been defined, the next stage in the development of an integrated system is the creation of algorithms to generate all the required sub-models for the various applications.
Figure 3.11: Overall structure of the unified model.
Chapter 4

Submodel Generation

Once the model structure has been defined, and all the necessary information has been obtained from the user, the information structure and associated algorithms must calculate all the other information it requires from the data that has been given. For example, processing times for each task are found by summing the processing times of the steps in that task.

The first section in this chapter describes this ‘post processing’, after which the generation of the individual models for the scheduling and supervisory control systems is detailed.

4.1 Information Post Processing

The purpose of post processing is to fill out the data structures with information which can be obtained from other parts of the data model.

The largest part of this deals with constructing the details of each task from the information given about the steps within each task. Additionally the tasks that may be performed by each unit, taking into account issues such as equipment states, where a unit may have to be in a certain state, such as ‘clean’, to perform a task, and limited connectivity are found.

4.1.1 Defining the suitability of each unit

When defining the units in a plant, a user does not specify the tasks which a unit can perform. For each recipe the resources used are defined, but gBSS requires that the tasks that may be performed by each unit be specified. This information could have been obtained directly from the user, but this would imply a repetition of the data, which opens up a possibility for the input to be inconsistent. Secondly by only defining resource usage in recipes, the user only has to ensure that the recipe definition is correct when adding or modifying a recipe.
In theory it should be quite simple to reverse the definition of unit usage given by the user, so that the units used by steps in a task perform that task. However the recipe may describe many resources, since it will describe the storage resources involved as well as process resources. Additionally there are some special cases such as where there is limited connectivity or continuous units.

Algorithm 4.1 shows the four level decision tree for assigning a task to the units which perform it, as follows:

1. Where there is limited connectivity, the user will have specified which unit(s) performs this task, as shown in Table 3.10. If there are such qualifiers in steps within a task then these units are the only units to perform these tasks and the assignment process is terminated.

2. If there is a continuous resource involved in a step in a task then this resource performs the task. Where there is a continuous unit in a process, it is typically taking material from one resource, processing it and transferring it to another resource. Therefore material passes through such a resource in the definition of the operation, as shown in Table 3.7. Usually such resources are part of the routing between two other resources, and would not normally be considered as candidates for performing the task. Therefore continuous resources are treated as a special case.

3. If there is a process step then the resource which is used performs this task. This is the normal assignment, since most tasks involve some processing of material within a resource.

4. If no resource has been found in steps 1 to 3 then the first transfer step in the task transfers material to the resource which performs the task. This catches the remaining tasks that do not involve any processing. All the steps must be transfers, so the initial destination for these transfers is assumed to perform this task.

Should this for any reason not result in the desired unit being marked as performing the task, the user may dictate a unit specifically in the same way as when there is limited connectivity (Table 3.10).

4.1.1.1 Modifying Equipment Suitability with Equipment State

Where there are restrictions on when a unit may perform a particular task, for example because it must be cleaned when changing from one product to another, preconditions will exist in the resource declaration for a task, as shown in step 1 in Table 3.9. These must be used to restrict the normal list of tasks a unit may perform, as shown in Algorithm 4.2.
Algorithm 4.1 (Determination of tasks performed by units).

FOR each task ...
1 IF no unit was specified as performing this task ...
   1.1 IF it uses a continuous resource ...
      1.1.1 Set the units in that resource to perform this task ...
   1.2 ELSE IF it contains a process step in a resource ...
      1.2.1 Set the units in that resource to perform this task ...
   1.3 ELSE set the units in the target of the 1st transfer to perform this task...

If a resource in the first step in a particular task contains a list of one or more pre-
conditions, then the first stage is to find out what the condition of the unit will be after
performing this task. This is found by looking at each of the other steps in this task, and
finding the final output state that leaves the resource which had the precondition.

Algorithm 4.2 (Restrictions on unit performance with state).

FOR each task ...
1 FOR each step ...
   1.1 FOR each PRECONDITION statement ...
      1.1.1 Find the last outstate from this task.
      1.1.2 FOR each unit ...
         1.1.2.1 IF the unit performs this task ...
            1.2.1.1.1 Task is only performed in state PRECONDITION.
            1.2.1.1.2 Unit changes to the outstate when performing task.

It is now known in which conditions a unit may perform this task, and what the
resulting condition will be. For each unit which performs this task, it is now restricted so
that it is only performed in the correct conditions, whereupon it switches to the resulting
condition.

4.1.1.2 Generation of task level details from step data

In the input language, the user has specified the names of the tasks, which recipes they
are in and which steps are in each task. The scheduling model will require information
about the states entering and leaving each task, when and in what amounts, common
resource utilisation over the task duration, as well as an overall duration for the task.
Firstly since we are examining each master recipe and task, the tasks are checked to find which are "first" tasks, in that they contain one or more steps that have no precedents, and which are "last" tasks, in that they contain one or more steps that are not precedents for any other step. The lists of first and last tasks will be used later when translating schedules in Chapter 5.

Secondly if transfers are modelled so that the transfer step appears in two tasks, any common resource utilisation associated with that step must only be attributed to one of the tasks. Therefore each step is marked so that it only counts for common resource utilisation in one task, except where there is a task with identical steps. This ensures it is correctly modelled where there are two identical tasks to overcome limited connectivity. Algorithm 4.3 shows how this is done.

The steps are searched in reverse order so that the tasks that are being transferred to will have the common resource attached. This ensures that if the first task releases a STABLE state which is stored in the unit in which it was produced, then the common resources associated with the transfer will only be used when that transfer finally takes place.

Algorithm 4.3 (Determination of resource usage in tasks).

```
FOR each master recipe ...
  1 FOR each step (looping in reverse order) ...
     1.1 resos_set_in_task = NULL
     1.2 FOR each task in the recipe ...
        1.2.1 IF the task contains this step ...
           1.2.1.1 IF resos_set_in_task == NULL ...
              1.2.1.1.1 Task will take resource usage from this step.
              1.2.1.1.2 resos_set_in_task = this task.
           1.2.1.2 ELSE ...
              1.2.1.2.1 IF task contains the same steps as resos_set_in_task ...
                 1.2.1.2.1.1 Task will take resource usage from this step.
```

The main process for generating the task details from the given data is outlined in Algorithm 4.4. For each task in the model the first step is to set the overall processing time to zero. The the "target" resource is found, which is the resource which contains one or more units which perform this task.

Each step in the task is then examined. If it is a transfer to the target resource then the resource description for the target resource in that step is examined and any input or output states are added as input or output states to this task. Input states are assumed
to enter at the beginning of the step, and therefore occur at the current overall processing
time for the task. Similarly output states are assumed to leave at the end of the step,
and occur at the current overall processing time plus the step duration.

The proportion for this input or output state is set to the corresponding transfer size.
This will be adjusted later into a real proportion. Where a step contains a single transfer,
finding this size is trivial. However when there are multiple transfers in a single step, it
is harder to match each input or output state to the correct transfer. To some extent the
second input state should correspond to the second transfer, but this cannot always the
case. Therefore the transfer which corresponds to being in the correct position, involves
the resource from/to which the state leaves/arrives and correctly arrives/departs from
that resource is chosen. The function which returns the transfer size returns the transfer
with the best score on these three criteria, so even if a transfer is out of position with the
state descriptions, then the fact that it involves the target resource and is moving in the
correct direction should mean that the correct value is returned.

Processing steps are also allowed to add input and output states, since a recipe may
take material to a tank, process it and then leave it in place, for example the preparation
of a CIP solution. The proportion is set to the current amount in the vessel.

The common resource usage is added to the task from the current overall processing
time to the end of the step, together with the use of any resource that is modelled as a
common resource at the higher scheduling level. When the step has been examined its
duration is added to the overall processing time for the task.

After all the steps in the task have been analysed, each input and output state propor-
tion is divided by the maximum amount of material present during the task. This ensures
that the correct proportions for input states and output states are given no matter how
the amounts of material present within a unit fluctuates during the task. For example
for each of the holdup profiles shown in Figure 4.1, the proportions must be found to be
as shown in Figure 4.2. Simply summing the amounts of material entering or leaving the
task fails to give the correct proportions in every case. In the second and third cases in
Figure 4.1, the maximum amount of material present is 100, although the total amount
added is 125. Therefore only dividing each transfer by 100 gives the correct proportions.
Additionally when material enters or leaves a task by staying in a vessel rather than by
a transfer this method holds fast while others fail.

Finally each time in the task is then rounded using the scheduling interval size and
rounding coefficients given. Where input or output states of the same state now coincide
because of the rounding they are amalgamated. For instance if a state WATER is an
input state to a task after 45 minutes and 75 minutes, and an interval size of one hour
with a rounding coefficient of 0.5 rounds both these input states to 60 minutes, then the
Algorithm 4.4 (Generation of Task details from detailed step information).

FOR each task ...
1 proc_time = 0.
2 amount_in_task = 0.
3 max_in_task = 0.
4 target_resource = resource with a unit that performs this task.
4 FOR each step in this task ...
4.1 IF it is a transfer to target_resource ...
4.1.1 FOR each instate in the step for this resource ...
4.1.1.1 Add an instate to this task at proc_time ...
4.1.1.2 Set the proportion of the instate to the corresponding transfer size.
4.1.1.3 amount_in_task += transfer size.
4.1.1.4 IF amount_in_task ≥ max_in_task ...
4.1.1.4.1 max_in_task = amount_in_task.
4.1.2 FOR each outstate in the step for this resource ...
4.1.2.1 Add an outstate to this task at proc_time + step duration.
4.1.2.2 Set the proportion of the outstate to the corresponding transfer size.
4.1.2.3 amount_in_task -= transfer size.
4.1.3 IF the resources in this step count for this task ...
4.1.3.1 FOR each resource in the step that is a higher level common resource ...
4.1.3.1.1 Add usage from proc_time to proc_time + step duration.
4.1.3.2 FOR each common resource used in the step ...
4.1.3.2.1 Add usage from proc_time to proc_time + step duration.
4.1.4 proc_time = proc_time + step duration.
4.2 ELSE IF it is a transfer from the target resource ...
4.2.1 FOR each instate in the step for this resource ...
4.2.1.1 Add an instates to this task at proc_time ...
4.2.1.2 Set the proportion of the instate to the corresponding transfer size.
4.2.1.3 amount_in_task += transfer size.
4.2.1.4 IF amount_in_task ≥ max_in_task ...
4.2.1.4.1 max_in_task = amount_in_task.
4.2.2 FOR each outstate in the step for this resource ...
4.2.2.1 IF proc_time = 0 ...
4.2.2.1.1 Add an outstate to this task at proc_time + step duration.
4.2.2.2 ELSE ...
4.2.2.2.1 Add an outstate to this task at proc_time.
4.2.2.2.3 Set the proportion of the out-state to the corresponding transfer size.
4.2.2.4 amount_in_task -= transfer size.
4.2.3 IF the resources in this step count for this task ...
4.2.3.1 FOR each resource in the step that is a higher level common resource ...
4.2.3.1.1 Add the usage from proc_time to proc_time + step duration.
4.2.3.2 FOR each common resource used in the step ...
4.2.3.3 Add the usage from proc_time to proc_time + step duration.
4.2.5 proc_time = proc_time + step duration.
continued on page 65 ...
4.3 ELSE ...

4.3.1 FOR each resource in the step that is a higher level common resource ...
   4.3.1.1 Add the usage to the task from proc_time to proc_time + step duration.
4.3.2 FOR each common resource used in the step ...
   4.3.2.1 Add the usage to the task from proc_time to proc_time + step duration.
4.3.3 proc_time = proc_time + step duration.
4.3.4 FOR each instate in the step for this resource ...
   4.3.4.1 Add an instate to this task at proc_time ...
   4.3.4.2 Set the proportion of the instate to the amount_in_task.
   4.3.4.3 amount_in_task = 0.
4.3.5 FOR each outstate in the step for this resource ...
   4.3.5.1 Add an outstate to this task at proc_time + step duration.
   4.3.5.2 Set the proportion of the outstate to the amount_in_task.
   4.3.5.3 amount_in_task = 0.
5 IF amount_in_task ...
   5.1 FOR each instate or outstate proportion ...
      5.1.1 Add -1 x amount_in_task.
6 FOR each instate or outstate proportion ...
   6.1 IF proportion == 0 ...
      6.1.1 proportion = 1.0.
   6.2 ELSE ...
      6.2.1 Divide proportions by max_in_task.
7 Round each time in the task to an exact number of intervals.
8 Amalgamate in or out states of the same state at the same time.
9 Remove any common resource usage rounded away.

proportions are summed and one is removed since gBSS will not allow more than one input state of the same state at the same time. Similarly resource usage is ignored if it is rounded away.

4.1.1.3 Find minimum and maximum sizes for each task

In SUPERBATCH, recipes are defined for a nominal batch size. Minimum and maximum sizes are then specified, within which the sizes of control recipes may lie. Each quantity in the recipe which varies with batch size is then adjusted according to Equation 4.1.

\[
Actual quantity used = Control Recipe Size \times \frac{Quantity in recipe}{Nominal Recipe Size}\]  \hspace{1cm} (4.1)

In gBSS, each task performed by each unit may be given a range of utilisation factors, specifying the proportion of the unit which must be used when that task is performed.
For example, a utilisation of 0.5:1.0 specifies that the unit must be at least half occupied when it performs that particular task. This information can be generated automatically for each task from the master recipe data.

This is of particular importance when cleaning tasks are included in the model, since if no minimum size is specified for the cleaning task then gBSS will reduce the cleaning batch size to zero in order to minimise any costs attached to either performing this task or the states produced by the cleaning task. Additionally the extra constraints these utilisations add to the MILP may lead to faster solutions, although this is not always found to be the case.

For each task, we look at the first transfer into the task and calculate the minimum and maximum sizes for this task as follows,

\[
\text{Minimum task size} = \frac{\text{Minimum Recipe Size}}{\text{Nominal Recipe Size}} \times \text{Maximum quantity in task} \quad (4.2)
\]
Maximum task size = \( \frac{Maximum \ Recipe \ Size}{Nominal \ Recipe \ Size} \times \text{Maximum quantity in task} \) (4.3)

If there are no input states to this task which are transfers, (for example if the task processes material left in a unit by another task), then the first transfer out of the task, and the corresponding output state proportion is used instead.

Each unit which performs this task then divides these minimum and maximum sizes by the unit size to give the required fractions.

4.1.1.4 Example of generating task details

As an example the task level details for the task task b.1 shown in Table 3.9 are found as follows.

To start with, the first step in this task has precondition statements, and therefore the units that perform this task must be restricted to these conditions. step 3 is the final step in this task, and Mixer.#1, which performs this task, has an output state of int b.1. Therefore Mixer.#1 performs task task b.1 when in condition detergent and switches to condition int b.1, and also when in condition int b.1 and switches to condition int b.1.

The first step, step 1 shows that this task has an input state of raw b after 2 minutes, adds two minutes to the overall processing time and adds 100 to the amount of material in the task. The input state is given a fraction of 100 for the moment.

step 2 adds 3 minutes to the overall processing time. step 3 shows that this task has an output state of int b.1 after 7 minutes, (the overall processing time now equals 5, plus this step duration of 2 minutes), and the fraction is again set equal to the transfer amount for the present. The 2 minutes are added to the overall processing time and 100 is removed from the amount of material in the task.

The maximum amount of material in the task at any time was 100, and so each of the input and output state proportions is divided by 100, which makes them both 1.0. The overall task details are therefore written in the gBSS input language as follows:

\[
\text{TASK task.b.1 PROC.TIME 7.0} \\
\text{INSTATE raw.b FEED.TIME 2.0} \\
\text{OUTSTATE int.b.1 PROC.TIME 7.0}
\]

Finally the minimum and maximum processing amounts for this task are found using Equations 4.2 and 4.3. The minimum, maximum and nominal recipe sizes are found in the recipe header, and for the this example are assumed to be 50, 100 and 100 respectively. The maximum amount of material present within this task is 100, and therefore the
minimum and maximum task sizes are 50 and 100. The unit Mixer.1 has a size of 100, and therefore the minimum and maximum proportions that will be written in the gBSS input file will be 0.5 and 1.0 as shown below.

UNIT Mixer.1
   CAPACITY 100.0
   PERFORM task.b.1 UTILISATION 0.5:1.0

4.1.2 Further Processing

In addition to examining the recipe information, certain other features must be added to the model. Firstly extra resources and recipes are added to the model so that orders, receipts and equipment breakdowns can be correctly modelled throughout the system.

Secondly the operations available have been given in terms of resources, (i.e. resource-operations), whereas they must also be specified in terms of units, (i.e. unit-operations), for on-line execution, and this may be done automatically.

4.1.2.1 Addition of supplier and customer resources

SUPERBATCH and the input language described in Chapter 3 can specify resources as feed or product, where there is an infinite amount of space or material available. Since storage may be a limiting factor in scheduling, all resources of type feed and product are labelled as being of type storage and two new resources, named SUPPLIER and CUSTOMER are added, and the flowsheet adjusted accordingly. SUPPLIER is a feed resource and CUSTOMER is a product resource. Master recipes are also added that make receipts from the SUPPLIER to the original feed resources, and deliveries to the CUSTOMER from the original product resources.

![Figure 4.3: External interactions to and from the site.](image-url)
This aligns the gBSS and SUPERBATCH models, so that the orders and deliveries in
the gBSS model correspond to transfers to and from the SUPPLIER and CUSTOMER
resources in the SUPERBATCH model. All units within the site are then of finite capacity,
and the situation is as shown in Figure 4.3. Where there actually is unlimited storage for
a material then it may still be simply modelled as being much larger than the other units
and therefore effectively infinite.

At the same time as the connection between the SUPPLIER and CUSTOMER re-
sources and the original feed and product resources are added, the connections between
individual units are found, since the input language only specified the connections between
resources. The individual unit connections are used when creating the unit operations,
described in Section 4.1.2.3, and in the generation of a simulation model, described in
Section 4.4.

4.1.2.2 Addition of breakdown recipes

SUPERBATCH does not have an internal way of representing unit unavailability. Instead a
batch must be added for a recipe which uses the unit for the duration of the breakdown.

Therefore for each resource a recipe is added, with the name resource.name.unavail,
and a parameter which is used as the duration for the single step within the recipe. The
duration of the breakdown can therefore be fixed when the breakdown is scheduled. The
step calls an operation, which also must be created for each resource, which simply carries
out a process in that resource.

4.1.2.3 Creation of unit operations

When the operations available within the plant were defined in Section 3.4, they were
defined in terms of resources. In the SUPERBATCH language this is a “resource-phase”.

When running off-line these “resource-phases” are sufficient for SUPERBATCH to create
a schedule. However when SUPERBATCH is running on-line, “unit-phases” are executed
in the control system, where a “unit-phase” is simply a phase defined in terms of units
rather than resources. Each resource-phase can exist as one or more unit-phases, where
the choice of units in each resource is specified. This allows the specification of particular
parameters for a phase depending on which units are used.

In a complete SUPERBATCH model, valid unit-phases are defined by the user. However
to speed implementation all possible unit phases are generated automatically for each
resource phase. Firstly a unit-phase is created for every possible combination of units in
the phase. Then these are checked against the flowsheet and any invalid phases removed
where transfers would otherwise pass between unconnected units.
4.1.2.4 Handling stable states

g*BSS handles stable states by automatically generating additional tasks with names derived from the state name in the form ST_stable_state_name. These tasks last for one time interval, consume the state at the beginning and produce it at the end. Since these tasks may appear in the g*BSS output they must appear in the overall model as well otherwise they will not be recognised.

Therefore for each stable output state in the tasks within a master recipe, a task is added to mirror the task which will be added by g*BSS. For example if there is a stable state INT, then the following task, (shown in the g*BSS input language), is added to each master recipe containing one or more tasks with the output state INT:

```plaintext
TASK ST.INT
    INSTATE INT
    OUTSTATE INT PROC.TIME 1.0
```

assuming that the interval size is 1 hour. These tasks are not written in the g*BSS model, since g*BSS includes them automatically.

4.2 Generation of Scheduling Models

This section describes the creation of off-line scheduling models. At this stage only the basic model is described. When running in a closed loop mode with an on-line supervisory package, the off-line model must be modified to take account of the current plant status. However these modifications are described separately in Chapter 6.

4.2.1 g*BSS Model

The g*BSS model consists of three files, a problem description file, a State Task Network file and a unit description file.

The problem description file details the information that is relevant to the current situation. The solution methodology, scheduling horizon, initial quantities of materials and details of the orders are given. When each order is first read from the user it is given a unique order number. This is passed to g*BSS so that individual orders may be identified in the results.

The State Task Network file firstly lists all the states. Then each task is given together with its input and output state timings and proportions and common resource usage. The unit description file details each unit, its size, the tasks which it may perform and the states which it may store.
4.2.2 Minimax Scheduling Model

The minimax scheduling algorithms described in Xia and Macchietto (1994) are run from a single input file, containing matrices representing the various facets of the problem. Such a format makes this file difficult for a user to write manually, but once the structure is identified then these models can be created from the overall model just as easily as a gBSS model.

The first step is to find the number of process resources and the number of product recipes, since these numbers define the size of the matrices used. Most of the data is then what is normally available, the number of units in each resource, the volume of each unit, the connectivity between the units, etc.

However this algorithm is not suitable for the following types of problem:

1. Situations where storage units may also perform tasks.
2. Situations where the product structure involves intermediates, since batch integrity is lost if material is mixed and the individual batches are no longer identifiable.
3. Situations where unit conditions limit the tasks a unit may perform. The cleaning times between products may be specified in the minimax model, however this requires cleaning tasks to be identified in the global model.
4. Situations involving common resources, (although the formulation could theoretically be extended to include these).

The objective used by the minimax algorithm is to maximise the rate of profit by finding the optimum number of batches of each product between given minimum and maximum numbers and their optimal order. This is difficult to integrate with the concept of an order driven system, so in the current implementation the range is set as between 1 and 3 batches of each product.

4.2.3 SA/GA Scheduling Model

The algorithms in the EASY package described in Xia and Macchietto (1997) are run from a single data file containing details of units and the STN in a similar format to gBSS and is written in a similar way.

However the nonlinear formulation allows the processing time to be given in the form:

\[
\text{Processing time} = \alpha + \beta \times \text{Batch size}^7
\]  

(4.1)

Again a number of extensions would be required to EASY to allow it to be fully integrated. These include:
1. Offset timings for input and output states, to allow a wider range of processes and also transfers to be modelled.

2. Receipts and deliveries during the scheduling horizon, for when these occur in practice, and to allow control recipes to be fixed.

3. Equipment unavailability, both for when this occurs due to breakdown etc., and to allow control recipes to be fixed as described in Chapter 6.

4. Common resources.

5. Unit states.

### 4.3 Generation of the Supervisory Control Model

The SUPERBATCH model consists of a single file detailing the materials involved, the units, their connectivity, the operations, and the master recipes. This information has all been defined by the user, (with the exception of the unit phases), and all that is required is for it to be written out in the format described in Appendix B. In addition an empty schedule file is written, detailing the initial inventories and equipment availability.

### 4.4 Generation of the Plant Simulation Model

The system developed by Liu (1995) linked the SUPERBATCH on-line monitor to a simulation model running in the gPROMS software package (Barton and Pantelides, 1991). The information is passed via a piece of software called ControlLink, as shown in Figure 4.4. The result is a system where schedules may be run replicating the behaviour that would occur when running on a real plant.

![Diagram](image-url)

**Figure 4.4:** Interactions between SUPERBATCH, ControlLink and gPROMS.

ControlLink requires the mapping data between the control system tags that SUPERBATCH will use and the variables in the gPROMS model. Since both of these are known
because they have both come from the single model, the mapping files can be automatically generated, and are guaranteed to be consistent and correct. This is in contrast to the manual generation of these files, which is tedious and prone to typographical errors.

The gPROMS model follows the format as described by Liu (1995). It starts with a definition of each unit, as shown in Table 4.1. Each unit has a variable for the holdup within the unit together with one for each flow entering and leaving the unit. The mass balance can then be written for each unit. The streams are linked according to the plant flowsheet.

Then each operation is described as shown in Table 4.2. When the operation is started, an iteration counter for the number of times that the operation has been performed, IT\text{\_}op\_name, is incremented, and the value of the local variable for this operation, L\text{\_}op\_name, is set to 1 so that the simulation knows that this operation is being performed.

The duration is calculated using the duration of a step which uses this operation, with an additional random quantity if necessary. Assuming that the duration is still positive then the operation simply continues for the required duration. Afterwards both the local and remote variables for this operation, L\text{\_}op\_name and R\text{\_}op\_name, as well as the ControlLink tag, are set to 0 so that both ControlLink and gPROMS know that the operation is complete.

A monitor task, part of which is shown in Table 4.3, monitors the variable for each operation using the GET command to obtain the variable from ControlLink, and when necessary starts the appropriate operation. The initiation and completion of each operation is also reported to the user as shown.

The model is therefore extremely basic, and could be extended to implement the mass balance. It could also be extended manually to include detailed unit models and controllers which could be used to determine operation durations, and introduce variations in the quantities executed. Development of the work by Barber (2000) could be incorporated into the data model to produce a detailed dynamic model from first principles from a detailed physical description.

A simulation model may also be part of the testing process for procedure synthesis (Baird, 1999), and therefore the possibility of reusing this model a number of times during plant and process design and operation exists.

4.4.1 Historical ‘play-back’ facility

In some situations it may be desirable to force the gPROMS model to replicate actual operations timings from historical data obtained from a plant, for example to compare the actions of an integrated system with those that were actually carried out.
MODEL T.4

PARAMETER
  Size AS REAL

VARIABLE
  Fin1 AS Flow_rate
  Fin2 AS Flow_rate
  Fout1 AS Flow_rate
  Fout2 AS Flow_rate
  Fout3 AS Flow_rate
  Holdup AS Volume

STREAM
  Input1 : Fin1 AS Process_Stream
  Input2 : Fin2 AS Process_Stream
  Output1 : Fout1 AS Process_Stream
  Output2 : Fout2 AS Process_Stream
  Output3 : Fout3 AS Process_Stream

EQUATION
  $Holdup = 0 + Fin1 + Fin2 - Fout1 - Fout2 - Fout3 ; # Mass Balance

END

Table 4.1: Example gPROMS model language for a unit operation.

TASK SPARGE.31

PARAMETER
  Plant AS MODEL Plant

VARIABLE
  ActualIrur AS REAL

SCHEDULE
  SEQUENCE
    RESET Plant.IT.SPARGE.31 := OL(T(Plant.IT.SPARGE.31) + 1); ENL
    RESET Plant.L.SPARGE.31 := 1.0; ENL
    ActualIrur := 9900 + NORMAL(0,990); ENL
    IF ActualIrur < 1 THEN
      ActualIrur := 1;
    ENL
    CONTINUE FOR ActualIrur
    RESET Plant.L.SPARGE.31 := 0.0; ENL
    RESET Plant.R.SPARGE.31 := 0.0; ENL
    SEND
      "SPARGE.31.state" := 0;
    ENL
    ENL
    ENL

Table 4.2: Example gPROMS model language for a unit operation.
Table 4.3: Example gPROMS model language for the monitor task

Table 4.4: Format of file for specifying historical data

Alternatively, it may be wished to repeat a set series of plant operations with a number of alternative rescheduling criteria. When the model is running normally the variations in timings are random, and therefore we might want to fix the timings with a certain variability.

To this end the facility to write exact timings for each occurrence of each unit operation is included. The first step is to specify timings for each unit operation in a file as shown in Table 4.4. The duration of each unit operation in minutes is taken from this file.

The 'Plant' section of the gPROMS model contains an iteration counter for each unit operation. Each unit operation 'TASK' then updates this iteration counter and contains IF statements so that the appropriate duration is used. Once all the given durations have been used, the model returns to using the nominal timings with the given variation.

4.5 Conclusions on model generation

The model described in Chapter 3 therefore allows the generation of a number of scheduling models, a supervisory control model and a basic simulation model. Alternative
scheduling models could also be written, for example that described by Zentner et al. (1998) in the Resource Constrained Scheduling Problem specification (RCSpEc) language. A large number of current scheduling publications, for example Rodrigues et al. (1999), Ierapetritou et al. (1999) and Orcun et al. (1999), are also based on the STN model, and it would therefore also be possible to create input for any of these applications. A standardised format would simplify the process of using different algorithms.

These models are guaranteed to be consistent, since they originate from the same data objects. Problems with mapping say, “Reactor_1” and “reactor-1”, because the two models were written by different people with different styles at different times, are eliminated.

To modify these models all that is required is a modification in the original data model, and then the new sub-models can be simply regenerated, and there is no doubt that all the relevant models have been changed in the relevant places.

The simulation model could be extended, either manually, or a more detailed model could be automatically generated (e.g. including mass balances and flows).

The various models for the applications that are to be integrated can now be created, and a hierarchical model structure exists. The final pieces required for an integrated system are therefore the routines to pass information from the off-line scheduling package to the on-line execution package and back again. The next two chapters describe these routines.
Chapter 5

Implementation of Off-line Schedules

This chapter describes how a schedule from an off-line scheduling package may be translated into a schedule that an on-line supervisory control package may implement. The principal software package used for the off-line scheduling has been the gBSS package, which was developed at Imperial College, but the techniques could be applied to any system that gives out a list of tasks at the times at which they should take place.

5.1 Sorting the gBSS schedule

gBSS gives as an output a list of tasks on each unit. This information can be simply reordered into an overall list of chronological tasks. Since we are more concerned with when batches finish, rather than when they begin, the list is reversed so that when we sweep through to find a list of batches, they are ordered as to when they finish.

This reverse chronological list of tasks, together with the time each begins, its size, and the unit in which it takes place, provides a suitable interface to the main translation algorithm, since any package using a scheduling model based on the tasks defined in the global overall model ought to be able to provide this information.

5.2 Creating a list of control recipes

Once a reverse chronological list of tasks has been created, the next step is to extract the 'control recipes', as defined in the S88.01 standard, (ISA, 1995). Basically this is the list of batches that must be executed.

The overall method is shown in Algorithm 5.1. The aim is to sweep through the reverse chronological list, looking for tasks which show the end of a control recipe. Once one of these 'last' tasks is found, the size of this control recipe may be calculated, and then all the other tasks that are part of this control recipe must also be found, since the control recipe may take place over a number of units, and these additional tasks may be required to give additional information such as the choice of units used to perform the earlier tasks.
Additionally, the ‘last’ tasks have been defined as tasks containing steps which are not preconditions to any other step in the master recipe, i.e. once that step has been done, part or all of the recipe is complete. However, if the recipe carries out any tasks in parallel, there may be more than one ‘last’ task in a particular recipe, and therefore all the tasks in the list associated with a recipe must be allocated so that one occurrence of a control recipe does not lead to multiple recipes in the translated schedule, with one occurrence for each of the last tasks.

Algorithm 5.1 (Creation of Control Recipes - Backwards Sweep).

FOR each unallocated task on the reverse chronological list ...  
1 IF it is a last task ...  
1.1 Calculate the size of the new control recipe.  
1.2 Create a list of steps which must be performed in this recipe.  
1.3 Create a list of tasks which must be examined, (initially this last task).  
1.4 WHILE there are tasks on the list to be examined ...  
1.4.1 FOR each task on this list ...  
1.4.1.1 FOR each instate to that task ...  
1.4.1.1.1 FOR each unallocated task on the chronological list ...  
1.4.1.1.1.1 IF an instate coincides with an outstate of the same state ...  
AND the steps in the new task have still to be done ...  
1.4.1.1.1.1.1 Allocate the new task as part of this control recipe.  
1.4.1.1.1.1.2 Remove the steps in this task from those yet to be done.  
1.4.1.1.1.1.3 Add the new task to the list to be examined.  
1.4.1.2 FOR each outstate from that task ...  
1.4.1.2.1 FOR each unallocated task on the chronological list ...  
1.4.1.2.1.1 IF an outstate coincides with an instate of the same state ...  
AND the steps in the new task have still to be done ...  
1.4.1.2.1.1.1 Allocate the new task as part of this control recipe.  
1.4.1.2.1.1.2 Remove the steps in this task from those yet to be done.  
1.4.1.2.1.1.3 Add the new task to the list to be examined.  
1.4.1.3 Remove this task from the list to be examined ...

Algorithm 5.1 shows how two tasks are found to be part of the same recipe. For a particular task under examination, another task may release a state of material at the same time as this task takes in the same material, in this case at time A, and therefore this preceding task is also part of the same control recipe. Similarly if another task takes in a state of material at the same time as this task releases the same material then again this subsequent task is also part of the same control recipe.  

An additional check is that the steps in the new task have not already been carried out by a task already allocated as being in this recipe. This situation might arise if a number
of parallel interconnected units were carrying out the same tasks at the same time, and therefore a number of identical tasks might fulfil the above criteria.

Once a particular task has been examined, then the tasks that have been found also to be part of the control recipe must also in turn be examined, and so on until no further tasks are found.

When the schedule has been created by a discrete time formulation then it might be more accurate to compare all timings in terms of the discrete intervals, but if continuous time formulations are used this may not be so simple, and therefore timings are compared in hours subject to a small tolerance. A value of 0.01 hours is used since gBSS rounds all times to two decimal places, and therefore this overcomes any problems when the interval size is a fraction of an hour, e.g. 1.33 and 1.34 are not overlooked as different times.

From Equation 4.2, which defined the lower limit on the task size, and replacing the minimum sizes with actual sizes, we obtain

$$Actual \ task \ size = \frac{Actual \ Recipe \ Size}{Nominal \ Recipe \ Size} \times Maximum \ quantity \ in \ task \quad (5.1)$$

and rearranging gives,

$$Actual \ Recipe \ Size = \frac{Actual \ Task \ Size \times Nominal \ Recipe \ Size}{Maximum \ quantity \ in \ task} \quad (5.2)$$

which is used to calculate the control recipe size from the last task size that has been found.

The rationale for sweeping backwards through the chronological list of tasks is that it is the time at which a batch finishes, rather than when it begins, that is of interest since the material cannot be shipped until it has been finished. Therefore the sequence in which the batches finish should be the order in which they are prioritised resources in SUPERBATCH. However, for comparison, the tasks may be searched in the other direction, as described in Algorithm 5.2. In this case the search is scanning initially for 'first tasks', i.e. tasks containing steps with no preceding steps. Once a 'first task' is found the process of assigning the other tasks in the control recipe is identical to the backwards sweep.

### 5.3 Ordering the list of control recipes

In the majority of cases, the order in which the control recipes are found is the order in which they can be put into SUPERBATCH. However in some cases, especially those involving limited intermediate storage, some reordering may be necessary.

This is because the discrete time formulation of gBSS, and in particular the fact that transfers take place instantaneously, may lead to intermediate materials being produced
Algorithm 5.2 (Creation of Control Recipes - Forwards Sweep).

FOR each unallocated task on the chronological list ...
  1 IF it is a first task ...
     1.1 Calculate the size of the new control recipe.
     1.2 Create a list of steps which must be performed in this recipe.
     1.3 Create a list of tasks which must be examined, (initially this last task).
     1.4 WHILE there are tasks on the list to be examined ...
        1.4.1 FOR each task on this list ...
           1.4.1.1 FOR each instate to that task ...
              1.4.1.1.1 IF an instate coincides with an outstate of the same state ...
                  AND the steps in the new task have still to be done ...
                    1.4.1.1.1.1 Allocate the new task as part of this control recipe.
                    1.4.1.1.1.2 Remove the steps in this task from those yet to be done.
                    1.4.1.1.1.3 Add the new task to the list to be examined.
              1.4.1.2 FOR each outstate from that task ...
                 1.4.1.2.1 FOR each unallocated task on the chronological list ...
                    1.4.1.2.1.1 IF an outstate coincides with an instate of the same state ...
                        AND the steps in the new task have still to be done ...
                          1.4.1.2.1.1.1 Allocate the new task as part of this control recipe.
                          1.4.1.2.1.1.2 Remove the steps in this task from those yet to be done.
                          1.4.1.2.1.1.3 Add the new task to the list to be examined.
                  1.4.1.3 Remove this task from the list to be examined ...
and consumed at the same instant. SUPERBATCH fully allocates resources to each control recipe before looking at the next one in the list, and therefore will fail to find a feasible schedule if an intermediate material requires storage space which is not made available until the next recipe is scheduled.

This problem is overcome by performing a simple mass balance on each control recipe as it is added to the final list that is given to SUPERBATCH. If a batch fails the mass balance by drawing on material that is not present, or by trying to store material when there is not room, it is skipped until it can be scheduled. This effectively reverses the order of pairs of batches where problems might arise.

This first step is to find the maximum storage available for each state, which was done when the overall model is originally processed. Then the initial amounts of each state must be found as follows:

1. current_storage for each state = 0.0
2. For every storage unit, if it contains a state at the start of the scheduling horizon add it to the current_storage for that state.
3. If a virtual order or receipt is made add or remove this amount from current_storage.

5.3.1 Translation of deliveries

The first stage in translating the orders is to read in the information in the gBSS output file about each order. The information required is the order name, the actual amount, and the actual time.

Where upper and lower times for an order are given, gBSS may make a number of shipments at a number of times to fulfill a single order. Therefore each amount shipped is treated as a separate control recipe which will be added to the list of recipes to be executed. The earliest starting time for the recipe is fixed as the earliest time at which the order may be fulfilled.

The control recipes from the deliveries are then added into the list of future control recipes at the appropriate position given its timing.

5.4 Specifying unit and material allocation in the control recipes

The master recipe definitions in SUPERBATCH are given in terms of resources and generic materials. When a control recipe is created, (i.e. an actual batch is scheduled), the exact unit in each resource and the specific material in each generic material must be specified.

\footnote{This is explained more fully in Chapter 6}
By sweeping through the list of translated tasks, for each task, the unit that the task takes place in defines a unit-resource pairing, since each unit is only a member of one resource. This pairing information is sent to the control recipe to see whether it adds to the current specification of which units are to be used.

Similarly, the input states and output states of each task uniquely define a specific material and a generic material, since each state is only a member of one specific material, and each specific material is only a member of one generic material. This pairing of generic and specific materials is again sent to the control recipe to see whether it adds to the current specification of which specific materials are to be used.

In some cases, notably where there are multiple storage vessels, the unit resource pairing is not found, since gBSS lumps all storage for a state together, and therefore does not specify which unit stores a given quantity of material. Where there is only one unit in the resource, this is not an issue, and the unit resource pairing can be given by default. By modelling each storage unit as a separate vessel, and therefore moving the STN description towards the maximal STN described by Crooks (1992), then problems with storage allocation are avoided.

However, this can lead to very large scheduling problems, which may restrict the use of the technology in situations where the problem is already close to the limits of what can be solved in a reasonable length of time. An alternative would be to introduce heuristics to determine the storage allocation, or to leave these decisions to plant operations personnel. SUPERBATCH already contains algorithms to give a default feasible allocation but at present these have to be authorised manually, but could easily be modified to allow a completely automatic system.

5.5 Example of schedule translation

Figure 5.2 shows the flowsheet of an example plant. The process consists of taking three raw materials, weighing them carefully, and adding them sequentially to the reactor. At various points during the reaction, three product streams are taken, two of which then undergo a packing operation. The STN, shown in Figure 5.3, is branched with a number of states entering and leaving the main reaction task during its execution. It therefore represents most of the difficulties in finding control recipes from the list of tasks.

In this case the "last" tasks are React, Can-off and Drum-off, since the steps transferring the states F, Cans and Drums at the end of these tasks are not precedents for any other step. The "first" tasks are the three weighing tasks, since these contain steps with no precedents.

The gBSS model is given the objective that the product has a value of 1.0, and therefore simply makes as many batches as possible. The resulting schedule from executing gBSS is
shown in the Gantt chart in Figure 5.4. The list of tasks is found from a gBSS output file and chronologically ordered by finishing time to give the list shown in Table 5.2, (except initially the right-hand column will be empty, since tasks will not have been allocated to control recipes).

When sweeping backwards through Table 5.2, executing Algorithm 5.1, line 24 is examined first, which corresponds to an unallocated last task. Therefore a new control recipe is created, the size of which is found using Equation 5.2,

\[
\text{Actual Recipe Size} = \frac{30 \times 100}{30} = 100
\]

From Table 5.1 it can be seen that Drum_off has an input state of E at time 0 in the task, which will be at the start time 6.1 plus 0 in the schedule. The next step is to search through the other lines in Table 5.2 for a task which has 30 units of E as an output state at time 6.1. Line 22 is for task React, which at a start time of 5.5 plus a processing time of 0.6, i.e. 6.1, has an output state of E, of the correct size.

Therefore line 22 is also allocated to control recipe IC_BATCH_1, and added to the list
Table 5.1: State Task Network for the example in tabular form

<table>
<thead>
<tr>
<th>Task</th>
<th>Processing time (hrs)</th>
<th>State</th>
<th>In/Out</th>
<th>Proportion</th>
<th>At time</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEIGH_A</td>
<td>0.5</td>
<td>A</td>
<td>In</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Out</td>
<td>1.0</td>
<td>0.4</td>
</tr>
<tr>
<td>WEIGH_B</td>
<td>0.7</td>
<td>B</td>
<td>In</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Out</td>
<td>1.0</td>
<td>0.6</td>
</tr>
<tr>
<td>WEIGH_C</td>
<td>0.8</td>
<td>C</td>
<td>In</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Out</td>
<td>1.0</td>
<td>0.8</td>
</tr>
<tr>
<td>REACT</td>
<td>1.1</td>
<td>WEIGHED_A</td>
<td>In</td>
<td>0.25</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WEIGHED_B</td>
<td>In</td>
<td>0.375</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WEIGHED_C</td>
<td>In</td>
<td>0.625</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>Out</td>
<td>0.25</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E</td>
<td>Out</td>
<td>0.375</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>Out</td>
<td>0.625</td>
<td>1.0</td>
</tr>
<tr>
<td>CAN_OFF</td>
<td>0.7</td>
<td>D</td>
<td>In</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CANS</td>
<td>Out</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>DRUM_OFF</td>
<td>0.9</td>
<td>E</td>
<td>In</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DRUMS</td>
<td>Out</td>
<td>1.0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Figure 5.4: gBSS Gantt chart for the Example Plant.
of tasks to have its input and output states examined. In this way lines 19 through to 24 are all allocated to IC\_BATCH\_1, and no further lines are found to be connected. Then line 18 is found to be a last task, a new control recipe is created, and the process is repeated as before.

Eventually four control recipes are created, and these can be passed through the SUPERBATCH planner to give the Gantt chart shown in Figure 5.5. In this case the mass balance on each recipe raised no problems, since each recipe is simply taking material from the raw material tanks where there is plenty of inventory, and placing material into the product tanks, where there is plenty of space, and therefore no reordering is required.

It should be noted that the SUPERBATCH schedule looks slightly different from the gBSS schedule, where the batches are spread out over the seven hour horizon. In the more detailed SUPERBATCH schedule all four batches are completed within five hours. This is because gBSS can only fit a maximum of four batches into the seven hour horizon, and therefore all solutions producing four batches of product are optimal according to the objective used, no matter how the slack time is spread through the schedule. The scheduling algorithm within SUPERBATCH minimizes the make-span by starting each operation

---

<table>
<thead>
<tr>
<th>Line No.</th>
<th>Unit</th>
<th>Task</th>
<th>Start time</th>
<th>End time</th>
<th>Task size</th>
<th>Control recipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WEIGH_VESSEL_A</td>
<td>WEIGH_A</td>
<td>1.1</td>
<td>1.6</td>
<td>20</td>
<td>IC_BATCH_4</td>
</tr>
<tr>
<td>2</td>
<td>WEIGH_VESSEL_B</td>
<td>WEIGH_B</td>
<td>1.1</td>
<td>1.8</td>
<td>30</td>
<td>IC_BATCH_4</td>
</tr>
<tr>
<td>3</td>
<td>WEIGH_VESSEL_C</td>
<td>WEIGH_C</td>
<td>1.1</td>
<td>1.9</td>
<td>50</td>
<td>IC_BATCH_4</td>
</tr>
<tr>
<td>4</td>
<td>REFLUX_REACTOR</td>
<td>REACT</td>
<td>1.5</td>
<td>2.6</td>
<td>80</td>
<td>IC_BATCH_4</td>
</tr>
<tr>
<td>5</td>
<td>CANNING_LINE</td>
<td>CAN_OFF</td>
<td>1.9</td>
<td>2.6</td>
<td>20</td>
<td>IC_BATCH_4</td>
</tr>
<tr>
<td>6</td>
<td>WEIGH_VESSEL_A</td>
<td>WEIGH_A</td>
<td>2.2</td>
<td>2.7</td>
<td>20</td>
<td>IC_BATCH_3</td>
</tr>
<tr>
<td>7</td>
<td>WEIGH_VESSEL_B</td>
<td>WEIGH_B</td>
<td>2.2</td>
<td>2.9</td>
<td>30</td>
<td>IC_BATCH_3</td>
</tr>
<tr>
<td>8</td>
<td>WEIGH_VESSEL_C</td>
<td>WEIGH_C</td>
<td>2.2</td>
<td>3</td>
<td>50</td>
<td>IC_BATCH_3</td>
</tr>
<tr>
<td>9</td>
<td>DRUMMING_LINE</td>
<td>DRUM_OFF</td>
<td>2.1</td>
<td>3</td>
<td>30</td>
<td>IC_BATCH_4</td>
</tr>
<tr>
<td>10</td>
<td>REFLUX_REACTOR</td>
<td>REACT</td>
<td>2.6</td>
<td>3.7</td>
<td>80</td>
<td>IC_BATCH_3</td>
</tr>
<tr>
<td>11</td>
<td>CANNING_LINE</td>
<td>CAN_OFF</td>
<td>3</td>
<td>3.7</td>
<td>20</td>
<td>IC_BATCH_3</td>
</tr>
<tr>
<td>12</td>
<td>WEIGH_VESSEL_A</td>
<td>WEIGH_A</td>
<td>3.3</td>
<td>3.8</td>
<td>20</td>
<td>IC_BATCH_2</td>
</tr>
<tr>
<td>13</td>
<td>WEIGH_VESSEL_B</td>
<td>WEIGH_B</td>
<td>3.3</td>
<td>4</td>
<td>30</td>
<td>IC_BATCH_2</td>
</tr>
<tr>
<td>14</td>
<td>WEIGH_VESSEL_C</td>
<td>WEIGH_C</td>
<td>3.3</td>
<td>4.1</td>
<td>50</td>
<td>IC_BATCH_2</td>
</tr>
<tr>
<td>15</td>
<td>DRUMMING_LINE</td>
<td>DRUM_OFF</td>
<td>3.2</td>
<td>4.1</td>
<td>30</td>
<td>IC_BATCH_3</td>
</tr>
<tr>
<td>16</td>
<td>REFLUX_REACTOR</td>
<td>REACT</td>
<td>3.7</td>
<td>4.8</td>
<td>80</td>
<td>IC_BATCH_2</td>
</tr>
<tr>
<td>17</td>
<td>CANNING_LINE</td>
<td>CAN_OFF</td>
<td>4.1</td>
<td>4.8</td>
<td>20</td>
<td>IC_BATCH_2</td>
</tr>
<tr>
<td>18</td>
<td>DRUMMING_LINE</td>
<td>DRUM_OFF</td>
<td>4.3</td>
<td>5.2</td>
<td>30</td>
<td>IC_BATCH_2</td>
</tr>
<tr>
<td>19</td>
<td>WEIGH_VESSEL_A</td>
<td>WEIGH_A</td>
<td>5.1</td>
<td>5.6</td>
<td>20</td>
<td>IC_BATCH_1</td>
</tr>
<tr>
<td>20</td>
<td>WEIGH_VESSEL_B</td>
<td>WEIGH_B</td>
<td>5.1</td>
<td>5.8</td>
<td>30</td>
<td>IC_BATCH_1</td>
</tr>
<tr>
<td>21</td>
<td>WEIGH_VESSEL_C</td>
<td>WEIGH_C</td>
<td>5.1</td>
<td>5.9</td>
<td>50</td>
<td>IC_BATCH_1</td>
</tr>
<tr>
<td>22</td>
<td>REFLUX_REACTOR</td>
<td>REACT</td>
<td>5.5</td>
<td>6.6</td>
<td>80</td>
<td>IC_BATCH_1</td>
</tr>
<tr>
<td>23</td>
<td>CANNING_LINE</td>
<td>CAN_OFF</td>
<td>5.9</td>
<td>6.6</td>
<td>20</td>
<td>IC_BATCH_1</td>
</tr>
<tr>
<td>24</td>
<td>DRUMMING_LINE</td>
<td>DRUM_OFF</td>
<td>6.1</td>
<td>7</td>
<td>30</td>
<td>IC_BATCH_1</td>
</tr>
</tbody>
</table>

Table 5.2: Chronological list of tasks for the Example Plant
as soon as is feasible, giving a neater and more efficient schedule.

The schedule was also translated using the forward sweeping algorithm, Algorithm 5.2, and in this case exactly the same result was found.

![Gantt chart](image)

**Figure 5.5: SUPERBATCH Gantt chart for the Example Plant.**

### 5.6 Example of schedule reordering

To demonstrate the need for reordering the translated schedule, the plant and processes shown in Figure 5.6 is considered. Two raw materials are reacted together to form an

![Flowsheet and State Task Network](image)

**Figure 5.6: Example plant and process where reordering may be necessary.**
intermediate which is then blended with one of two additives to form a product. It is the limited storage available for the intermediate that causes the problem.

When the overall data model is written and a gBSS model generated and executed, the schedule shown in Figure 5.7 is generated. The two products were simply given prices, and gBSS has found a schedule which maximises the production of the more valuable product.

![Figure 5.7: gBSS Gantt chart for the Example Plant.](image)

There is a simple one to one mapping between the tasks shown in the STN and the master recipes, and therefore the actual translation is fairly trivial. The result is a list of recipes in the same order as the chronological tasks in the gBSS schedule. This is shown in Table 5.3. However, if the amount of the intermediate C present after each recipe has been executed is calculated, it can be seen that there is often more C present than the 420 units which may be stored in the intermediate tank. Therefore this schedule will fail if it is executed in SUPERBATCH.

Therefore the mass balance described in Section 5.3 is required. Only those recipes that do not violate the mass balance for any state may be added to the list to be executed. In this case control recipe 1 is feasible, but control recipe 2 violates the storage constraint on state C. Therefore it is ignored for the moment, and control recipe 3 is added next. Again recipe 2 is tested, but there is still a violation so recipes 4, and then 5 are added, until eventually recipe 2 is feasible. This process continues through the entire schedule, until the list becomes that shown in Table 5.4.
### Table 5.3: Initial order of control recipes.

<table>
<thead>
<tr>
<th>Control Recipe Number</th>
<th>Master Recipe</th>
<th>Amount of C remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>REACTION</td>
<td>320.00</td>
</tr>
<tr>
<td>2</td>
<td>REACTION</td>
<td>640.00</td>
</tr>
<tr>
<td>3</td>
<td>BLENDING.2</td>
<td>546.25</td>
</tr>
<tr>
<td>4</td>
<td>BLENDING.2</td>
<td>452.50</td>
</tr>
<tr>
<td>5</td>
<td>BLENDING.2</td>
<td>358.75</td>
</tr>
<tr>
<td>6</td>
<td>REACTION</td>
<td>678.75</td>
</tr>
<tr>
<td>7</td>
<td>BLENDING.2</td>
<td>585.00</td>
</tr>
<tr>
<td>8</td>
<td>BLENDING.2</td>
<td>491.25</td>
</tr>
<tr>
<td>9</td>
<td>BLENDING.2</td>
<td>397.50</td>
</tr>
<tr>
<td>10</td>
<td>REACTION</td>
<td>717.50</td>
</tr>
<tr>
<td>11</td>
<td>BLENDING.2</td>
<td>623.75</td>
</tr>
<tr>
<td>12</td>
<td>BLENDING.2</td>
<td>530.00</td>
</tr>
<tr>
<td>13</td>
<td>BLENDING.2</td>
<td>436.25</td>
</tr>
<tr>
<td>14</td>
<td>REACTION</td>
<td>756.25</td>
</tr>
<tr>
<td>15</td>
<td>BLENDING.2</td>
<td>662.50</td>
</tr>
<tr>
<td>16</td>
<td>BLENDING.2</td>
<td>508.75</td>
</tr>
<tr>
<td>17</td>
<td>BLENDING.2</td>
<td>475.00</td>
</tr>
<tr>
<td>18</td>
<td>BLENDING.2</td>
<td>381.25</td>
</tr>
<tr>
<td>19</td>
<td>BLENDING.2</td>
<td>287.50</td>
</tr>
<tr>
<td>20</td>
<td>BLENDING.2</td>
<td>193.75</td>
</tr>
</tbody>
</table>

### Table 5.4: Reordered list of control recipes.

<table>
<thead>
<tr>
<th>Control Recipe Number</th>
<th>Master Recipe</th>
<th>Amount of C remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>REACTION</td>
<td>320.00</td>
</tr>
<tr>
<td>3</td>
<td>BLENDING.2</td>
<td>226.25</td>
</tr>
<tr>
<td>4</td>
<td>BLENDING.2</td>
<td>132.50</td>
</tr>
<tr>
<td>5</td>
<td>BLENDING.2</td>
<td>38.75</td>
</tr>
<tr>
<td>2</td>
<td>REACTION</td>
<td>358.75</td>
</tr>
<tr>
<td>7</td>
<td>BLENDING.2</td>
<td>265.00</td>
</tr>
<tr>
<td>8</td>
<td>BLENDING.2</td>
<td>171.25</td>
</tr>
<tr>
<td>9</td>
<td>BLENDING.2</td>
<td>77.50</td>
</tr>
<tr>
<td>6</td>
<td>REACTION</td>
<td>397.50</td>
</tr>
<tr>
<td>11</td>
<td>BLENDING.2</td>
<td>303.75</td>
</tr>
<tr>
<td>12</td>
<td>BLENDING.2</td>
<td>210.00</td>
</tr>
<tr>
<td>13</td>
<td>BLENDING.2</td>
<td>116.25</td>
</tr>
<tr>
<td>15</td>
<td>BLENDING.2</td>
<td>22.50</td>
</tr>
<tr>
<td>10</td>
<td>REACTION</td>
<td>342.50</td>
</tr>
<tr>
<td>16</td>
<td>BLENDING.2</td>
<td>248.75</td>
</tr>
<tr>
<td>17</td>
<td>BLENDING.2</td>
<td>155.00</td>
</tr>
<tr>
<td>18</td>
<td>BLENDING.2</td>
<td>61.25</td>
</tr>
<tr>
<td>14</td>
<td>REACTION</td>
<td>381.25</td>
</tr>
<tr>
<td>19</td>
<td>BLENDING.2</td>
<td>287.50</td>
</tr>
<tr>
<td>20</td>
<td>BLENDING.2</td>
<td>193.75</td>
</tr>
</tbody>
</table>
The overall mass balance is identical, with the same amount of intermediate remaining at the end. All the control recipes found from the gBSS schedule should eventually be scheduled feasibly, since the mass balance in the underlying MILP model ensures this. Should any recipe fail the mass balance then this is a sign that something has gone wrong in the translation process.

The list shown in Table 5.4 is scheduled by SUPERBATCH without any errors, and results in the SUPERBATCH schedule shown in Figure 5.8.

![Figure 5.8: SUPERBATCH Gantt chart for the Example Plant.](image)

### 5.7 Translation from Minimax and SA/GA packages

The schedules obtained from the Minimax and EASY models generated in Sections 4.2.2 and 4.2.3 can also be translated in a similar manner.

The EASY package generates a list of tasks for each unit and this is read and ordered in exactly the same way as the gBSS output, only the format is different. The fact that the result is from a continuous time formulation with exact task timings does not affect the translation at all.

One feature of the continuous time formulation is that the user specifies the number of event nodes for the horizon. This seems to have a huge effect on the solution time, or
whether indeed a solution is found at all. Whereas the discrete time granularity can be
decided upon with a reasonable degree of certainty beforehand, the number of nodes for
the continuous time formulation was found more by trial and error.

The Minimax model was based on complete recipes rather than tasks and the results
are given as an ordered list of recipes, and therefore no translation as such is required.
This list can again be read from the output text files, which are actually in a format for
input to an earlier version of SUPERBATCH, and then these recipes can be written straight
to a SUPERBATCH input file.

In general, the scheduling results from these stochastic search based packages were
not as good as those found by gBSS, although this may simply reflect the much greater
development effort which has gone into gBSS.

5.8 Conclusions from schedule translation

This chapter has described an algorithm for translating schedules obtained from off-line
optimal scheduling packages into a list of batches, which can then be entered into a
supervisory control package.

Two variations of the basic algorithm are proposed, but these will have to be compared
to see if there is any variation in performance.

The incorporation of the Minimax and EASY packages shows that the methods devel-
oped in this chapter are not specifically based around the gBSS package, and in theory
any scheduling tool which outputs either a list of tasks, their timings, sizes, and equip-
ment usage, (which includes all formulations based on the STN), or a list of recipes to be
executed, could also be integrated into the system.

The barrier to using a different tool is therefore writing the code to read in the result
in the format that the tool uses. The three tools mentioned here were all developed at
the same department, and yet each uses a different format. If the format for schedule
output were standardised then scheduling packages could be switched at will.

At the time of writing, the ISA S88.02 standard seems to be trending towards all
information being stored in a Structured Query Language (SQL) database, and if this were
extended to include scheduling as well as control information then this would represent
just such a standardised format.
Chapter 6

Rescheduling with the Current Plant Status

Zentner et al. (1994) state that “Ideally, scheduling and planning tools are best deployed when relevant information is automatically accessible through electronic means.” Part of this relevant information is the current status of operations and inventories within the plant itself, and this chapter presents a methodology for automatically providing an optimal off-line scheduling package with this information.

The supervisory control system provides this information, which not only can be used to initialise off-line rescheduling, but may also be analysed so that the integrated system can respond automatically to situations where it is desirable to initiate off-line rescheduling.

The first part of this chapter details what information is read from the supervisory system and how it is stored, so that the plant status is consistent at any moment in time. The transition from the on-line system to an off-line system also involves moving from a continuous time representation to a discrete time representation (at least for some scheduling algorithms) and therefore this needs to be taken into account. Then the way in which the current plant status is passed to the scheduling package is described.

6.1 Reading the current on-line schedule

SUPERBATCH represents the expected availabilities and activities of plant resources using Resource/Unit Availability ProfileS, (RUAPS) and the theoretical development of these is described in Cott (1989). RUAPS are arrays indicating the expected points in time where the activity of a unit or resource changes, together with the new level of usage after the change. SUPERBATCH can be instructed to write a file containing RUAPS for every unit and common resource whenever the planner module is run, or else on demand through a link to the on-line monitor.

An example of the RUAPS is shown in Figure 6.1. Material is first transferred from Unit_A to Unit_B, then a process is carried out in Unit_B, and finally material is transferred from Unit_B to Unit_C. The events are as follows:
Chapter 6. Rescheduling with the Current Plant Status

- At time A - The first transfer begins.

- Between time A and time B - Material flows from Unit_A to Unit_B.

- At time B - The first transfer ends.

- At time C - The second transfer begins.

- Between time C and time D - Material flows from Unit_B to Unit_C.

- At time D - The second transfer ends.

This representation requires a continuous time format, since at any time between time A and time B, or time C and time D, the amounts are changing within the units. If such information is to be used in a discrete time model, such as that used by gBSS, then this information must be stored in such a way that the transfers occur instantaneously. The representation could be described as follows:

- Before time A - The material is in Unit_A.

- Between time A and time C - The material is used by a task, which will deliver material at time C, and Unit_B is in use until time D.

- From time C - The material is in Unit_C, and Unit_B is in use until time D.

This representation consistently represents the material location at any time, and is shown by the dashed line in Figure 6.1. It is the fact that the STN, (and RTN), formulation requires events to occur at discrete points in time that requires this shift in representation. Even continuous time formulations require events to occur instantaneously, and there is no consideration of a unit half way through filling, for instance. Rescheduling will always be initiated with the plant at the end of some events and at the beginning of others, even when events are occurring continuously on the real plant.

When scheduling the transfer of material SUPERBATCH can schedule the start of each transfer in a variety of ways dependent upon a user given parameter called the "optimistic timing" for each unit. This can vary between 1.0, which is the least optimistic, and 0.0, the most optimistic. For an optimistic timing setting of X, filling operations will be able to start once the spare capacity in a tank exceeds X x the capacity of the tank. Emptying operations will be able to start once the inventory exceeds X x the capacity of the tank. More optimistic timing results in better plant utilisation where there is buffer storage, but can lead to more interruptions when plant performance deviates from the nominal. There is also very occasionally a problem when optimistic operations for buffer tanks are converted to a representation with instantaneous transfers.
Figure 6.1: Example of RUAPS.

<table>
<thead>
<tr>
<th>RUAP column</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>Unit name</td>
<td>Read as the unit</td>
</tr>
<tr>
<td>uno</td>
<td>Unit number</td>
<td>Ignored</td>
</tr>
<tr>
<td>evtime</td>
<td>Event time in secs since 1970</td>
<td>Read as event time</td>
</tr>
<tr>
<td>entimest</td>
<td>Event time as YY/MM/DD HH:MM:SS</td>
<td>Ignored</td>
</tr>
<tr>
<td>evtype</td>
<td>Event type as number</td>
<td>Switch for what to do with data</td>
</tr>
<tr>
<td>evname</td>
<td>Event type as string</td>
<td>Ignored</td>
</tr>
<tr>
<td>forecast</td>
<td>1 if event is forecast</td>
<td>Ignored</td>
</tr>
<tr>
<td>cr</td>
<td>Control recipe name</td>
<td>Read as control recipe</td>
</tr>
<tr>
<td>step</td>
<td>Recipe step name</td>
<td>Read as step</td>
</tr>
<tr>
<td>crunitphase</td>
<td>Control recipe unit phase no.</td>
<td>Ignored</td>
</tr>
<tr>
<td>unitphaseno</td>
<td>Unit phase no.</td>
<td>Ignored</td>
</tr>
<tr>
<td>resource</td>
<td>Resource name</td>
<td>Ignored</td>
</tr>
<tr>
<td>capstep</td>
<td>1 = capacity step change</td>
<td>Ignored</td>
</tr>
<tr>
<td>cap</td>
<td>Capacity</td>
<td>Ignored</td>
</tr>
<tr>
<td>invstep</td>
<td>1 = inventory step change</td>
<td>Ignored</td>
</tr>
<tr>
<td>invlevel</td>
<td>Inventory</td>
<td>Read as unit inventory</td>
</tr>
<tr>
<td>mat</td>
<td>Material</td>
<td>Read as material name</td>
</tr>
<tr>
<td>clnlevel</td>
<td>Clean level</td>
<td>Ignored</td>
</tr>
<tr>
<td>unlock</td>
<td>Unit lock</td>
<td>Ignored</td>
</tr>
<tr>
<td>duration</td>
<td>Duration of phase</td>
<td>Read as step duration</td>
</tr>
<tr>
<td>invincr</td>
<td>Inventory change for phase</td>
<td>Ignored</td>
</tr>
</tbody>
</table>

Table 6.1: Action taken on each column in the RUAPS file.
Chapter 6. Rescheduling with the Current Plant Status

Figure 6.2 shows this possibility. If a short operation draws on material while a tank is being filled, with Unit \( Y \) in this case being an intermediate storage tank, then there will effectively be a negative amount being stored, as shown by the dashed line.

![Diagram of Unit X, Unit Y, and Unit Z showing potential pitfalls of instantaneous transfers.](image)

(a) Unit has negative inventory.  
(b) Unit exceeds capacity.

Figure 6.2: Potential pitfalls of instantaneous transfers.

This problem will occur whenever transfers are considered as instantaneous events, and is possible whether transfers are assumed to occur at the beginning or end of the actual transfer operation.

Therefore when representing the current situation in gBSS, care must be taken whenever a unit has either a negative inventory or an inventory exceeding its capacity. This is dealt with in Section 6.3.

6.2 The timing of rescheduling

Full optimal off-line rescheduling is costly computationally, and is an off-line process because it cannot be done quickly enough to be carried out on-line, unlike the on-line scheduling of SUPERBATCH, where many of the degrees of freedom have been fixed. Therefore the fact that the plant will carry on functioning while off-line scheduling takes place needs to be taken into account.

This is done using an "inviolable period", which is an arbitrary time given to the system by the user, which is hopefully just slightly longer than the time required to reschedule off-line and then make the appropriate changes on-line.

For example consider a situation where scheduling is likely to take 30 minutes, and a batch is due to finish in 20 minutes, after which a new batch will start. If the new batch is not fixed in the new schedule, then attempts to delete it will fail and the new schedule...
will not be adopted. Even if this were overcome by checking batches have not started before attempting to delete them, then adjusting the schedule with the new batches around a situation different from that assumed by the scheduling package may lead to highly suboptimal operation. An inviolate period of 30 minutes overcomes this problem. Depending on the variability of the process, 40 or 50 minutes might be preferable if the process might run faster (or if the off-line scheduling might be slower) than expected during the rescheduling.

The inviolate period also gives a handle on how much of the current schedule can be modified by off-line rescheduling and how much is to be left to on-line rescheduling. For example, if operators always need to know the schedule for the next six hours, and do not want this to change, then a six hour inviolate period will ensure that the product sequence is fixed for this horizon.

There is however a penalty for having a long inviolate period and that is that the flexibility of the schedule is reduced, and the performance of the system in response to a changing scenario is likely to be degraded.

6.3 Fixing Control Recipes

Each control recipe has estimated starting and finishing times, found from the timings of the earliest and latest steps of that recipe in the detailed on-line schedule.

An estimated starting time prior to the current time implies that this batch has already started. Therefore all recipes with an estimated starting time before now plus the inviolate period are designated as fixed control recipes.

The overall process for fixing control recipes is shown in Algorithm 6.1, each stage of which is now discussed in turn.

6.3.1 Making equipment unavailable

For each fixed control recipe, each unit is examined and it is made unavailable for the time it is performing this control recipe.

In the SUPERBATCH detailed schedule, storage units are marked as performing transfer steps to and from them. However in the gBSS model, only the unit that actually performs the task should be made unavailable for this period. For example when a feed storage unit transfers material to a reactor, if the storage unit is made unavailable then the gBSS model will need to find another unit to store the feed material while it is unavailable.

Therefore a unit is only made unavailable if it can perform a task in the master recipe this fixed control recipe represents. This still allows for units which may be used both for storage and to perform tasks.
Algorithm 6.1 (Fixing Control Recipes).

1 FOR each fixed control recipe ...
   1.1 FOR each unit ...
      1.1.1 Make it unavailable when this recipe is performed.
   1.2 FOR each step in the master recipe ...
      1.2.1 Find the unit which performs this step.
      1.2.2 For each instate ...
         1.2.2.1 If the instate can go to this unit ...
            1.2.2.1.1 If the original unit has over capacity at now - 1 sec ...  
            1.2.2.1.1.1 Add a delivery - the over capacity at this time.
            1.2.2.1.2 else if the original unit has negative capacity ...  
            1.2.2.1.2.1 Add a delivery - the negative capacity at this time.
            1.2.2.1.3 else ...  
            1.2.2.1.3.1 Add a delivery at this time.
         1.2.3 For each outstate ...
            1.2.3.1 If the outstate can come from this unit ...
               1.2.3.1.1 If the original unit has over capacity ...  
               1.2.3.1.1.1 Add a receipt - the over capacity at this time.
               1.2.3.1.2 else if the original unit has negative capacity at now - 1 sec ...  
               1.2.3.1.2.1 Add a receipt - the negative capacity at this time.
               1.2.3.1.3 else ...  
               1.2.3.1.3.1 Add a receipt at this time.

In the situation in Figure 6.1, the SUPERBATCH schedule states that Unit_A performs the transfer step between times A and B. Unit_B also performs the transfer step between A and B. Between C and D it performs the processing step, and between C and D it performs the second transfer step. Unit_C also performs the second transfer step between C and D. If Unit_A and Unit_C are storage units then the gBSS model will only require that Unit_B is made unavailable from time A to D. On the other hand if this is only a small part of the overall recipe, and other processes have been carried out in Unit_A and will be in Unit_C, then all three units will be performing tasks in the recipe and will be made unavailable in the gBSS model.

If Unit_A and Unit_C perform processing steps in another control recipe, then their availability is decided when that recipe is fixed. If they carry out another processing task and are made unavailable, then the material will appear as a 'virtual' receipt, as described in Section 6.3.3.

Since the times at which a unit is unavailable are taken directly from the detailed SUPERBATCH schedule, rather than from nominal durations in the model, any additional
information from the process, such as predictions for the duration of a step based on samples, are also reflected in the information passed to gBSS.

### 6.3.2 Reducing common resource availability

Each common resource starts with a profile of its availability. For each fixed control recipe any events in the list of actual events in the plant are used to reduce the profile of availability.

Then when the scheduling model is created, the actual availability of the common resource, rather than simply its maximum level, is written.

### 6.3.3 Adding 'virtual' receipts and deliveries

When control recipes are modelled as taking place by making the equipment unavailable, the amounts of material produced and consumed by these recipes must also be modelled. These amounts are therefore modelled as additional (pseudo) receipts and deliveries, on top of those which occur from suppliers and to customers. These receipts and deliveries have no value and are for fixed amounts at the fixed times when the material is produced or consumed by the recipe. In this way they must take place, and do not affect the objective function of the scheduling problem. They are guaranteed to be feasible in as much as the SUPERBATCH schedule has been calculated on the basis that the material is available.

If a step has input states, then there is a delivery of that state at the beginning of that step. If a step has output states, then there is a receipt at the beginning of that step. It is at the beginning since it must coincide with any input states to other tasks, and additionally, the storage must be available from the beginning of the step. The only problems this might cause is if gBSS wants to make a delivery of this material at this time, but in any case the difference in timing should be small.

The size of the delivery is given by

\[
Size = \frac{\text{Size of transfer}}{\sum \text{All transfers for this step}} \times \text{Size of operation} \quad (6.1)
\]

or the size of the inventory present, in cases where there is an output state but no transfers.

It is at this stage that any problems with negative or excessive inventories are dealt with, as shown in Algorithm 6.1. This procedure adjusts the size of the delivery or receipt to take into account the fact that the initial amount present within the storage unit will be altered so that the initial position is feasible. In practice this situation was found to only occur on a very small number of occasions.
6.4 Setting initial amounts

If a unit is holding an amount of material and it has not been made unavailable at the beginning of the schedule by the process described in Section 6.3.1, then this initial amount should be written to the scheduling model. No unit can therefore have both an initial amount and be committed to doing something at the beginning of the schedule, which would probably result in an infeasible problem for gBSS, since no solution exists.

This is obviously the case for storage units, when they will only be unavailable if they also can perform a task and are currently performing that task. Process units will usually either be empty or performing a task, and therefore will not have an initial amount. However if a separate master recipe creates a stable state that is stored in the unit that produced it, and this material is then used by another master recipe, this material will be written as an initial amount if it is being stored in a process unit, since that unit will not have been made unavailable.

Therefore both storage and process units are consistently handled whether they are performing steps, empty, or storing an amount of material.

Should the situation occur where a unit has a negative inventory at a given time then the inventory is taken as zero, (which means no initial inventory is written). The reduced amount of material is taken into account in the ‘virtual’ receipt written. Similarly, if the inventory in a unit exceeds its capacity, the unit is given an initial inventory equal to its capacity, and the receipts and deliveries are reduced to take account of this.

6.5 Preparing the mass balance

The mass balance used to ensure that the order of batches being passed to SUPERBATCH is feasible, described in Section 5.3, needs to be initialised correctly. This is carried out when the plant situation is written to gBSS. If multiple scheduling algorithms were running in parallel on different machines then the initial values would have to be kept for the translation of each schedule, or else the initialisation repeated before translation.

First the current amount stored for each state is set to zero. Then for each storage unit, if it contains a state at the start of the horizon then that amount is added for that state. When tasks are fixed in the scheduling horizon and ‘virtual’ orders and receipts are added to mimic their effects, these amounts adjust the current amount in storage for that state.

6.6 An example of rescheduling with a fixed recipe

For an example of rescheduling we take the example on-line Gantt chart shown in Figure 6.3. In this schedule there is a single batch being executed. The process consists of
Chapter 6. Rescheduling with the Current Plant Status

Figure 6.3: Initial detailed on-line schedule.

Figure 6.4: Resulting initial position in gBSS.
charging two materials to a mixer, and after mixing the material is transferred to a reactor where an additive is added and a reaction takes place. As can be seen in Figure 6.3, the second material is currently being charged to the mixer.

If rescheduling is initiated, the first step is to find the start of the horizon in the $gBSS$ model, by adding on the user-defined inviolate period to the current time, as shown in Figure 6.4. The control recipe shown has obviously started, and therefore is to be fixed. The first step is to make the equipment unavailable in the $gBSS$ model when it is actually in use. Therefore the Mixer and the Reactor will be unavailable for the periods shown in Figure 6.4 by parallel lines. The availability of steam will be reduced to that shown by the dark line, as the fixed usage is cut out of the full availability.

The mixing task has an output state when the material is about to be transferred and this is modelled as a receipt of material at this time, shown by the gold arrow. At the same time there is a delivery for the same quantity of that state to go into the reactor, shown by the purple arrow. Similarly, the additive is required as a delivery at the beginning of that transfer. Finally the product leaving the reactor is modelled as a further receipt at the beginning of the transfer out of the reactor.

6.7 Testing the consistency of rescheduling

To be certain that the rescheduling process is robust and consistent, the following test can be carried out.

An initial schedule is generated and implemented in SUPERBATCH, with a price given to each of the products, so the objective is to make as much of the most valuable products as possible. $gBSS$ models are then created and executed assuming rescheduling was triggered at each and every second of the current scheduling horizon, and the resulting objective function is obtained from $gBSS$.

So long as the initial raw material is constrained so that it is fully consumed before the end of the horizon, and the margin used in the MILP solver is set to 0%, so that only the globally optimum solution is returned, then the objective function should be constant. The same amount of product will always be made, and therefore this demonstrates that the rescheduling approach always keeps track of all material at all times.

This lengthy process, for example involving 36,000 executions of $gBSS$ to cover a ten hour scheduling horizon, has been successfully demonstrated on a number of example plants, and was a useful test for debugging the implementation of these algorithms.
6.8 Conclusions from rescheduling

In this chapter an algorithm for initialising gBSS with the current plant status has been proposed, and a methodology for proving its consistency has demonstrated its robustness.

The approach has focused on using existing scheduling package facilities, such as making equipment unavailable and inserting orders and deliveries over the event horizon. It can therefore be applied to any scheduling package with these features, and does not require modification to internal scheduling algorithms. This may however raise problems with some specialised scheduling formulations. For example that of Ierapetritou and Floudas (1998) requires special extensions simply to allow orders for materials during the horizon (Ierapetritou et al., 1999), rather than simply at the end of the horizon.

However by making equipment unavailable for periods of time, the scheduling problem is still constrained, and the improvements in scheduling efficiency through the reduction in the number of integer variables as reported by Chua (1995) are maintained.

A general model and algorithms have now been described which allow off-line schedules to be automatically implemented on-line for execution, and enables the off-line schedule to be based around an accurate up to date representation of the current plant status. This represents the basis for closing the loop between off-line scheduling and supervisory control, replicating the results of Chua (1995), but achieving this in a generic, systematic and flexible manner.

The next step is therefore to study the dynamics of this closed loop system, for which a robust implementation of the model and algorithms is required, and this is described in the next chapter.
Chapter 7

Prototype Implementation of the System

In order to fulfill a decision support role, the model structure, language and algorithms outlined in the previous chapters need to be implemented. To this end, a prototype implementation was developed in the C++ programming language (Stroustrup, 1991), called gBOSS++, a general Batch Operations Support Structure.

C++ allows the hierarchical data structures to be modelled in an object orientated fashion, taking into account some of the ideas and issues raised in the S88.02 standard, (ISA, 1997).

The user can interact with the program through a graphical user interface, (GUI), shown in Figure 7.1. This console allows the user to read in data models, start and link various additional pieces of software, and also control the flow of information between them.

![Figure 7.1: Prototype 'front end' on an integrated decision support system.](image)

This chapter starts by describing the various stages in operating the software. Then some of the issues in handling the information present within the system are described, along with the way in which gBOSS++ communicates with the on-line SUPERBATCH monitor. Finally some features which should be implemented in a more sophisticated system are described.
7.1 Operating the prototype system

The operation of the system can be split into three phases. Firstly the information and data model must be entered, then the various pieces of the integrated system must be pieced together before lastly information must be managed between the various applications.

In a real operating situation, these phases may be repeated whenever the information model changes, for example when a new recipe is to be introduced. Each phase is now considered in turn.

7.1.1 Data entry

The information is entered by means of text files, using the format described in Chapter 3. Three text files complete the description.

First an equipment definition file defines the resources, units and common resources, together with the unit flowsheet. Secondly, an actions file describes the materials, operations and recipes. Finally a situation description file specifies which equipment and action files are to be used, the initial amounts and orders for each state, together with information about scheduling options and parameters for the integrated system.

This format mirrors that of gBSS, and allows the same equipment and recipe descriptions to be used in a number of different scenarios. Once gBOSS++ is started, a situation file can be selected and is automatically read and processed as described in Chapter 4.

The input files are checked for consistency in the names used in the various parts of the model, e.g. that resources used by an operation have been defined, but most of the checking is left to the gBSS and SUPERBATCH packages once the sub-models have been generated. A more sophisticated system would incorporate these checks initially, but for the present they are not repeated.

7.1.2 Building and connecting the extended system

Once the data model is in place, the various applications and connections may be set up. The first step is to create all the model files required by each of the applications. This is also described in Chapter 4.

Then the system shown in Figure 7.2 can be created. gBOSS++, (item 1), is already in place and gBSS and the SUPERBATCH off-line planner, (items 2 and 3), are simply executed from the model files. The SUPERBATCH off-line planner allows a snapshot of the current schedule to be modified with proposed changes and then presented to a user to show what the outcome will be if the proposed schedule is passed on-line.
Chapter 7. Prototype Implementation of the System

Figure 7.2: Application interactions in the integrated environment.
The SUPERBATCH monitor, (item 4), is the core of the SUPERBATCH on-line system, monitoring the execution of the schedule and adjusting the plan as necessary. Figure 7.3 shows the main display of the monitor. This ‘on-line’ Gantt chart is similar to that produced by the planner, except that the vertical dashed line represents the current time and once a minute the chart is redrawn with the current situation. The monitor is connected to the gPROMS plant model, (item 6), through the ControlLink software, (item 5). A separate package, the ControlLink clock (item 7), is used to give each of the applications a single common reference time. This also allows the running speed of the system to be manipulated, for example so that long scenarios may be run through at three times real time.

![Figure 7.3: SUPERBATCH on-line Gantt chart.](image)

Before the system is set up, gBOSS++ obtains the time from the system clock. Once a SUPERBATCH monitor is started, gBOSS++ links to this server application using the RPC client/server software routines (Bloomer, 1991), and obtains the current time from there. This ensures that gBOSS++ is always using the same time-frame as SUPERBATCH, whether the ControlLink clock is providing the time or a real control system. This link to the SUPERBATCH monitor server allows gBOSS++ to call a number of SUPERBATCH functions, not only to obtain the current time, but also to write the current detailed schedule to a named file, and also to read an updated schedule file containing batches to be added and deleted.
Finally, a SUPERBATCH operator console (item 8), allows a user to obtain a printout of the current schedule, and generally allows a direct operator interface with the on-line system. This interface is completely independent of the gBOSS++ interface with SUPERBATCH, allowing SUPERBATCH to be operated as normal even when gBOSS++ is running.

Each of these packages may be run on separate machines, if required. This allows the scheduling package to be run on a dedicated processor, while the smaller loads of the other applications can be distributed around smaller less expensive machines.

Currently the system is set up to start and link to a new SUPERBATCH monitor. However it could just as easily link to an existing monitor server currently running. This would require gBOSS++ to recognise new control recipes in the detailed schedule obtained from the existing SUPERBATCH monitor, but all the required information, such as the units used, will be available in that schedule. Such routines would also allow manual schedules passed by a user directly to SUPERBATCH to be recognised by gBOSS++.

### 7.1.3 Controlling Information flow

Once a data model is present, and all the necessary parts of the system are running, the flow of information can begin. This can take place in a manual mode, where all exchanges take place in response to manual requests, or in an automatic mode where the current situation is analysed once a minute and appropriate action can then be taken. However in automatic mode, a user can still make the same manual requests which are responded to immediately.

#### 7.1.3.1 Manual mode

In manual mode the user controls the execution of applications and the flow of information. After generating the models, the user may run gBSS, read the results when it has finished, start SUPERBATCH and then read in the detailed schedule, using the buttons shown on the left of the interface shown in Figure 7.1.

The SUPERBATCH planner may be run in two different ways depending on whether there is a connection to a SUPERBATCH monitor or not. When there is a connection, the monitor is directed to write a snapshot of the current monitor position. The planner is then run from that snapshot, with the new schedule added as a list of changes to the snapshot. This ensures that the schedule obtained in the planner is as close as possible to what will be obtained when that schedule is passed to the on-line monitor, and takes account of all variations from nominal performance that have occurred.

However where there is no connection to a monitor the planner is run with the original SUPERBATCH plant model and a list of all the fixed batches in the schedule together with
the new ones. This allows gBOSS++ to be run quickly without the full on-line setup, for testing, debugging, etc.

The user can also specify which scheduling algorithm to use, from a choice of the standard gBSS MILP, or the stochastic based Minimax or EASY packages.

7.1.3.2 Automatic Mode

In automatic mode the current detailed schedule is obtained every minute from the Superbatch monitor. This is then analysed and if certain rescheduling criteria are met, a gBSS model reflecting the current position is written and gBSS executed. When gBSS has finished the result is automatically and immediately read and passed through the Superbatch planner. Whether the actual on-line schedule is updated is also decided upon according to some user-set criteria.

These criteria for rescheduling and updating the on-line schedule are described in Chapter 8.

7.2 Schedule Information Handling

Whether in manual or automatic mode, the repeated passing of schedules gives rise to large flows of information. This must be handled in a robust and systematic manner in order for the system to operate with integrity.

The schedules held within gBOSS++ consist of two parts. Firstly, there is a list of control recipes. These are held as two separate lists, a 'fixed' list and a 'future' list, depending on whether they may be altered by the scheduling package. The control recipes obtained from translating the gBSS scheduling results are always added to the 'future' list. When the expected starting time for a 'future' control recipe is found to be before the current time plus the inviolate period then the control recipe is moved onto the end of the 'fixed' list. These two lists form part of the schedule object shown in Figure 3.11 as part of the overall data model.

Once the list of control recipes has been passed to Superbatch, the detailed schedule, or 'RUAPS', (Resource/Unit Availability ProfileS), can be read for this schedule. These detail the timings of the operations carried out by each unit, and also the amounts stored, together with the utilisation of each common resource. This information is stored with the unit or common resource to which it applies. These are the 'profile' objects shown in Figure 3.11. They consist of a vector of availability, a vector of operations performed and a vector of materials stored. It is from these vectors that gBOSS++ takes the information to provide initial positions for rescheduling in gBSS, as described in Chapter 6.
Chapter 7.  Prototype Implementation of the System

The current implementation of gBOSS++ holds three schedules, named "wannabe", "pretender" and "actual". (More could be easily added, and would be particularly necessary if parallel processors allowed multiple schedules to be generated concurrently. This would result in an array of schedules in the place of the pretender schedule.) There are therefore three schedule objects holding lists of control recipes, and three attachments of vectors to each unit and common resource. These are used as follows:

- **wannabe** holds the latest schedule that has been generated by gBSS.
- **pretender** holds the next best schedule that gBOSS++ has found so far. This allows a schedule to be read but not adopted as the on-line schedule, and then kept perhaps to be adopted later.
- **actual** holds the actual on-line schedule.

The schedules may then be transferred from wannabe to pretender, and from pretender to actual.

### 7.2.1 Initial Start-Up

Initially when gBOSS++ is started, all schedules are empty, as shown in Table 7.1. If there is a SUPERBATCH monitor running then the RUAPS information will be read into the actual schedule, since it will have been started with an empty schedule (sched0).

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Control recipe list</th>
<th>Detailed schedule information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fixed</td>
<td>Future</td>
</tr>
<tr>
<td>wannabe</td>
<td>empty</td>
<td>empty</td>
</tr>
<tr>
<td>pretender</td>
<td>empty</td>
<td>empty</td>
</tr>
<tr>
<td>actual</td>
<td>empty</td>
<td>empty</td>
</tr>
</tbody>
</table>

Table 7.1: Initial Schedules.

Firstly gBSS is run, and the schedule is translated into a list of control recipes and placed in the wannabe schedule. At this stage no operations have begun, and therefore all the control recipes will be 'future' control recipes. This list of recipes is passed to the SUPERBATCH planner, which generates the detailed schedule. These are again read into the wannabe schedule to give the position shown in Table 7.2, where sched1 refers to information from this schedule.

If the criterion for moving the wannabe schedule to the pretender schedule is met, such as it has a larger objective function value, the position becomes that shown in Table 7.3. This criterion will be discussed in the next chapter.
Chapter 7. Prototype Implementation of the System

### Schedule Control recipe list

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Control recipe list</th>
<th>Detailed schedule information</th>
</tr>
</thead>
<tbody>
<tr>
<td>wannabe</td>
<td>Fixed: empty</td>
<td>schedl</td>
</tr>
<tr>
<td>pretender</td>
<td>Future: empty schedl</td>
<td>empty (sched0 from monitor)</td>
</tr>
<tr>
<td>actual</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.2: Schedules after initial gBSS run.

Again if the criteria to move the pretender schedule to being the actual on-line schedule are met, the position becomes that shown in Table 7.4. The RUAPS data from the planner is moved to the actual schedule but it is then overwritten by data coming from the SUPERBATCH monitor. If no monitor is running then the schedl data from the planner remains in place and is used as a basis for rescheduling.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Control recipe list</th>
<th>Detailed schedule information</th>
</tr>
</thead>
<tbody>
<tr>
<td>wannabe</td>
<td>Fixed: empty</td>
<td>empty</td>
</tr>
<tr>
<td>pretender</td>
<td>Future: schedl</td>
<td>schedl</td>
</tr>
<tr>
<td>actual</td>
<td></td>
<td>empty (sched0 from monitor)</td>
</tr>
</tbody>
</table>

Table 7.3: Schedules after promoting wannabe schedule to pretender.

At this stage all control recipes are still ‘future’ control recipes, and the SUPERBATCH monitor begins executing the actual schedule.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Control recipe list</th>
<th>Detailed schedule information</th>
</tr>
</thead>
<tbody>
<tr>
<td>wannabe</td>
<td>Fixed: empty</td>
<td>empty</td>
</tr>
<tr>
<td>pretender</td>
<td>Future: schedl</td>
<td>schedl from monitor</td>
</tr>
<tr>
<td>actual</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.4: Schedules after promoting pretender schedule to actual.

### 7.2.2 Closed Loop Operation

At some point in time, rescheduling may be initiated. The first step of this is to look through the list of ‘future’ control recipes in the actual schedule, and move any that have started, or are about to start within the inviolate period, into the list of ‘fixed’ control recipes. These ‘fixed’ control recipes are placed in each of the three schedules, since these batches will take place whatever changes are made to the rest of the schedule. The actual RUAPS are used to place initial inventories and current equipment actions into gBSS.
Chapter 7. Prototype Implementation of the System

The schedule position is then as shown in Table 7.5, with schedfl representing the fixed part of schedl, and schedx1 the future part.

Once gBSS has finished rescheduling, the schedule is translated into a new list of ‘future’ control recipes, (each with a unique name), and becomes the ‘future’ control recipes in wannabe. At this point a snapshot is taken from the monitor, and a change of schedule file is written, deleting the future control recipes from the actual schedule and adding the future control recipes from the wannabe schedule. The planner is run with the snapshot and the schedule file and the detailed information is stored in the wannabe detailed information, as shown in Figure 7.6.

Table 7.5: Schedules after starting rescheduling in gBSS.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Control recipe list</th>
<th>Detailed schedule information</th>
</tr>
</thead>
<tbody>
<tr>
<td>wannabe</td>
<td>schedf1 empty</td>
<td>empty</td>
</tr>
<tr>
<td>pretender</td>
<td>schedf1 empty</td>
<td>empty</td>
</tr>
<tr>
<td>actual</td>
<td>schedf1 schedx1</td>
<td>sched1 from monitor</td>
</tr>
</tbody>
</table>

Table 7.6: Schedules after reading the results of rescheduling.

If no monitor is running, the schedf1 and sched2 recipes are then run from scratch in the SUPERBATCH planner, to give a reasonable idea of how the new schedule would appear on-line. As before, if the schedule is promoted to the pretender and then to the actual schedule, the position will be as shown in Table 7.7.

Table 7.7: Schedules after rescheduling.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Control recipe list</th>
<th>Detailed schedule information</th>
</tr>
</thead>
<tbody>
<tr>
<td>wannabe</td>
<td>schedf1 empty</td>
<td>empty</td>
</tr>
<tr>
<td>pretender</td>
<td>schedf1 empty</td>
<td>empty</td>
</tr>
<tr>
<td>actual</td>
<td>schedf1 sched2 from monitor</td>
<td>sched1 + sched2 from monitor</td>
</tr>
</tbody>
</table>

If, for instance, the pretender schedule is not moved on-line then it is left in place, and any future schedules obtained from gBSS by rescheduling can be compared against
it, while the original actual schedule continues to be implemented. If at a later date a user wishes to force the promotion of the pretender schedule, this can be done but if the monitor rejects the schedule because some batches that are to be changed have already started, then the actual schedule is left in place. This is also the case if off-line scheduling takes too long and the new schedule is out of date, or if it is simply infeasible for any reason when an attempt is made to adopt it on-line.

7.3 Interface with SUPERBATCH

In order to interact with the online SUPERBATCH monitor, gBOSS++ connects to the SUPERBATCH server using RPC software (Bloomer, 1991). gBOSS++ attaches itself as a client to the SUPERBATCH monitor server, in a similar manner to the operator interface. This allows gBOSS++ to change the on-line schedule, request the current RUAPS file, a snapshot or the current time, automatically from the monitor without any requirement for action by the user.

7.4 Enhancements to the prototype system

At present there is no error checking as such in gBOSS++. Since gBSS, gPROMS and SUPERBATCH all check their models any problems come to light when these packages are executed, but a more robust system would catch these errors sooner.

If alternative manual schedules are implemented, then these can be passed into the wannabe schedule in a similar way to how the gBSS schedules are translated, and then passed through the system in the same way.

To a large extent the information model within gBOSS++ is static. In order to add a new recipe the original input files must be modified and then the system started from scratch. It would however be possible to modify the model while the system is running, and regenerate the models for the other applications. gBSS is run from scratch each time it is executed, so there would be no problem in simply writing new model files. The SUPERBATCH on-line monitor however might need extensions to allow on-line model additions.

Another useful extension would be the specification of a pre-existing gPROMS model, which may for example exist through work on the automatic generation of procedural control code (Baird, 1999), in which case the user could also be aided in the construction of the mapping of operations for ControlLink. The gPROMS model could be written with more mass balance information, since this information is available in the common model, which would allow this model to be used as a basis for procedural control work if there was no pre-existing model. This opens up the possibility of using the common model as a basis for design, allowing capacity studies and control code generation to be
closely tied with the detailed design of unit operations, or else opens up the possibility for considerable model reuse during the design and operation of a plant.

Finally there is the issue of how to remove historic information, since if run for any length of time this builds up in the system. This could easily be written to history files or a third-party database and then removed from memory, and again work in the S88.02 standard, (ISA, 1997), may result in a common standard format for this.

7.5 Conclusions for the prototype implementation

This chapter has described the implementation of the model structure and translation algorithms, which together with gBSS and SUPERBATCH, form a completely integrated prototype decision support system for the operation of flexible plants.

The successful operation of this system verifies the algorithms described in previous chapters, and proves that the model is complete. The system now provides a platform for investigating the criteria for initiating rescheduling, which will be discussed in the next chapter.
So far a model and a set of algorithms have been described which allow off-line schedules to be automatically created from an initial position consistent with the plant status, and then automatically translated and applied on-line.

This opens up the possibility of studying triggers for this process. In addition ways are required to measure the comparative benefits of different modes of operation. The first part of this chapter looks at a number of triggers for rescheduling that have been implemented within the gBOSS++ system described in Chapter 7. Then the decision whether to adopt the new schedule is studied, and a performance measure for the system is described so that some comparison of different operating policies can be made. Finally an initial attempt at an interface to the higher level planning function is described.

8.1 Rescheduling decision primitives

When the integrated system is running in an automatic mode, it reads the current on-line schedule every minute, as described in Section 7.1.3.2. It is at this point that decisions can be made as to whether off-line rescheduling is started.

A flag is used to denote whether gBSS is running, and is set to true whenever gBSS is started and false whenever gBSS exits. This is used to ensure that only one gBSS calculation is running at any one time, and this prevents the processor becoming bogged down with a number of concurrent jobs. At present, if the situation changes to the extent where the previous execution of gBSS is no longer valid then it must be stopped manually, or else it continues until finished.

The schedule read from the on-line SUPERBATCH monitor can be analysed for the following features, and the user can then select which of these rules should trigger rescheduling.

8.1.1 Decay in the objective function

The schedules obtained from gBSS have been developed so as to maximise an objective function, so it would seem reasonable to monitor this value in the detailed schedule.
Whenever the on-line schedule is updated, the objective function for that schedule over the next scheduling horizon is calculated from the on-line schedule information and recorded, using Equation 8.1.

\[
\text{Objective function} = \left[ \sum_{\text{all states}} P_s I_s + \sum_{\text{all orders}} V_i Q_i \right]_{\text{now+ sched horizon}} - \left[ \sum_{\text{all states}} P_s I_s + \sum_{\text{all orders}} V_i Q_i \right]_{\text{now}}
\]  

where \( P_s \) is the price of state \( s \), \( I_s \) is the current inventory of state \( s \), \( V_i \) is the value of order \( i \) and \( Q_i \) is the quantity delivered in order \( i \) by the current time. These are not all the terms that gBSS can use in its objective function, but these are the usual ones, and further items, such as the cost of common resource usage, could easily be added to the system. The values are calculated at the current time and also at the end of the scheduling horizon projected into the future from the current time.

When the on-line schedule is then sampled and the current objective function is calculated, it can be compared with the original value calculated just after the current schedule was transferred on-line. Initially all of the steps that add to the objective function lie in between the current time and the end of the scheduling horizon\(^1\), and therefore only appear in the left hand term of Equation 8.1. As the schedule is executed these steps are carried out and then their value appears in both terms, and the objective function falls. If it has fallen below a given percentage of the original value then off-line rescheduling can be triggered.

\[\text{Figure 8.1: Typical Objective Function Plot.}\]

This policy has the advantage that it ensures that the plant always has something to do, since the completion of recipes is driving rescheduling. However it does not respond to delays in processing, since the on-line schedule will be pushed out by SUPERBATCH, and the objective function will not fall. Typically this policy results in reasonably periodic

\(^1\text{Unless they have been pushed out beyond that because the off-line model is optimistic.}\)
scheduling, the objective function varying with a saw tooth pattern as shown in Figure 8.1. Here a plant with set prices for its products is run with the criteria to reschedule whenever the objective function falls to 50% of the initial value.

Whenever this occurs a new schedule is created and the objective value rises again to a maximum value. The addition of orders with widely differing values, or large departures from nominal operating times in the plant, simply make the tooth pattern less regular.

8.1.2 Every x minutes

The simplest rule is to allow the schedule to be updated at regular intervals. The effect is fairly similar to waiting for the objective function to fall but it is not disturbed by fluctuations in orders or the operation. However it may lead to inflexibility if rescheduling is started prematurely, and a disturbance occurs shortly after rescheduling begins. Therefore the period must be set so that scheduling is not too frequent, but frequent enough so that opportunities to improve the schedule are not missed.

The main advantage of periodic rescheduling is that it does not rely on any deviation from normal operation, or any other event other than the passing of time.

8.1.3 New order or receipt notified

Whenever the system receives notice of a new order or receipt, rescheduling can be triggered automatically. This makes sense since the current on-line schedule will have been made without any knowledge of this new order or receipt, and therefore even if the new order cannot be met, the decision not to do so should be made on the basis that this is not optimal, rather than a wish not to disturb the current schedule.

However, to make trade-offs between orders, the relative values should take into account the full economic impact of the order. For example it might be worth doing everything possible to please a large regular customer, which might not be reflected in the actual accounting value of the order.

New orders and receipts may be added manually through the gBOSS++ user interface, as shown in Figure 8.2, or they may appear having been entered in the initial input files as being placed at a later date as discussed in Section 3.5.1. Ideally orders should arrive automatically from other systems in the enterprise, such as a higher level planning system. Section 8.4 discusses a potential interface for this.

8.1.4 New order or receipt falls within the scheduling horizon

One problem with a fixed horizon scheduling tool is that orders which are just beyond the horizon are ignored. To try and combat this, they can be included as a “soft” order. i.e.
as a range of amounts with zero as a minimum, with a due date at the end of the horizon. For example there might be a very large order for a product, but if it only considered when it falls within the scheduling horizon it cannot be fully fulfilled. If it is considered beforehand as a soft order, then spare capacity can be used to start fulfilling it if possible. However this may not always be either optimal or desirable.

As an alternative, or possibly as an additional measure, scheduling can be triggered whenever an order or receipt falls within the horizon when previously it was outside. This allows the schedule to reflect this order in its true form as soon as possible.

8.1.5 Order delayed beyond the due date

The actual completion time for each order is obtained from the on-line schedule, and this can be compared with the actual time the delivery must be made. If during the course of the execution of a schedule, the control recipe representing the delivery becomes delayed by more than a given tolerance then rescheduling can be triggered. A tolerance is given since a one minute delay might not be significant, whereas twenty minutes is. In practice such delays can be immediately costly if delivery trucks, or railway wagons etc. are kept waiting.

This policy makes full use of the ‘look ahead’ capability offered by on-line scheduling systems such as SUPERBATCH, allowing the effects of variations in plant performance to be seen over the entire schedule in real-time. Without on-line scheduling the effects of a number of small variations may not be noticed until it is too late to do anything about it.
However if it is still too late to do anything about it, for example if the schedule is very tight and a major breakdown occurs, then it is likely that the resulting gBSS problem will be infeasible (assuming the order has a minimum amount which must be made). This situation may require that the impossible order is either removed from the order slate or “softened” by reducing the lower bound, probably manually or with complex heuristics, otherwise no new schedules will come from gBSS until the infeasible due date has passed and the order is no longer considered.

8.1.6 Disruption coefficient

A disruption coefficient can be defined as follows,

\[
\text{Disruption Coefficient} = \frac{1}{n} \sum_{i=0}^{n} (|F_{\text{est},i} - F_{\text{orig},i}| \times D_{MR,i})
\]  

(8.2)

where \( n \) is the number of batches in the schedule that have not finished, \( F_{\text{est},i} \) is the current estimated finishing time for batch \( i \), \( F_{\text{orig},i} \) is the original estimated finishing time for batch \( i \), which was found immediately after the current schedule was adopted on-line.

\( D_{MR,i} \) is a disruption factor for the master recipe that batch \( i \) represents. This allows recipes such as cleaning to be removed from the disruption coefficient, by setting \( D_{MR,i} \) equal to zero, or for particularly important recipes to be weighted more heavily. As a default all master recipes start with \( D_{MR,i} = 1 \).

The absolute magnitude of the disruption is used so that the overall movement away from the original schedule is found. Otherwise large deviations could occur, but they could cancel each other out if some are delays and others early completions. This coefficient treats all movement away from the original plan as a potential for improving upon the current executing schedule. It would however be trivial to implement other measures with different treatments of deviations, e.g. ignoring early completions.

8.1.7 Equipment breakdown

Whenever a new equipment breakdown is notified rescheduling can be triggered. At present SUPERBATCH implements breakdowns as a separate fictional recipe (SUPERBATCH User Manual, 1996), such as CHILLER.OUT.OF.SERVICE, that is scheduled with the other recipes, but has a fixed starting time and the duration is a parameter set when the recipe is scheduled. Such recipes can be added through the gBOSS++ interface as shown in Figure 8.3, and rescheduling can be automatically triggered whenever such a breakdown is added.
8.2 Updating the on-line schedule

Once a new schedule has been created and translated, a decision has to be made as to whether to update the current on-line or next best schedules with this new schedule.

Three alternatives are given for this decision,

- The schedule can always be promoted without question.
- The schedule is promoted if the objective function is better than the existing schedule.
- The schedule is promoted if the user accepts it.

The user also has the option of viewing the alternative lists of recipes and promoting them whenever desired, as shown in Figure 8.4. With further programming effort this could be the basis for allowing the user to change the schedule manually, (e.g. delete a batch), or adjust batch parameters, (e.g. reduce the size of a schedule).
Figure 8.4: Display of control recipe lists.
8.3 A performance measure for an integrated system

In order to compare various rescheduling and update policies, some measure of how well the plant is being run is required.

Since the scheduling is being optimised to maximise an objective function it makes sense to base the performance measure on similar criteria. In the same way that the objective function is calculated over the next scheduling horizon, an amount of added value can be calculated over the length of time the system has been run for, as shown in Equation 8.3.

\[
\text{Added value} = \left[ \sum_{\text{all states}} P_s I_s + \sum_{\text{all orders}} V_i Q_i \right]_{\text{now}} - \left[ \sum_{\text{all states}} P_s I_s + \sum_{\text{all orders}} V_i Q_i \right]_{\text{initial position}} \tag{8.3}
\]

This is similar to Equation 8.1 except that the values are calculated at the current time and at the initial time of starting the system, and therefore is effectively an objective function of what has been achieved while the system is running.

The added value is tracked over time, together with the objective function for each schedule held and the disruption coefficient, and can be viewed as shown below in Figure 8.5. Further items, such as utility costs, could also be added in a similar manner. Additional measures of performance, such as those described by Lionis (1997), could also be calculated. Some of the analysis described requires further parameters, such as equipment capital cost, which could easily be added to the overall data model, but most is based on the RUAPS data file, and could therefore be straightforwardly calculated.

8.4 Interface to a planning system

The methods for interfacing an integrated plant decision support system to wider business planning functions is a complete area of research in itself. However an initial opening to such an interface could be provided as follows.

The gBOSS++ system checks a file called weekly.production each minute. This file has the structure shown in Table 8.1. The file states which week in the year this set of orders is for. The orders for each state at each site are listed, together with a due date and an amount. (EoW stands for End of Week). When this file is read, only the orders for the relevant site are picked up. If they are not already present within the list of orders for that state held within gBOSS++, then they are added, and if rescheduling is triggered by adding new orders then it is started.

This allows a single planning function to generate the production requirements for a number of sites, and for each of these sites to then automatically obtain those orders
Figure 8.5: Display of performance measures.
which pertain to them.

8.5 Conclusions on system operation and performance

Given the vast range of possible plants, recipes and scenarios it is difficult to describe a policy or policies that will provide good performance in every case.

However it is highly likely that any implementation would use some periodic rescheduling decision primitive as a base and then pick and choose from the rest as experience and circumstance dictated, and a wide range have been presented in this chapter.

A commercial concern is unlikely to allow the actual plant schedule to be modified without human authorisation unless the system proved itself to be incredibly reliable. In theory the schedule returned from rescheduling should always be at least as good, if not better, than the current schedule, so user confirmation seems the most likely option for schedule promotion.

The availability of a second processor would open up the possibility of starting gBSS again if further triggers occur during rescheduling. Different algorithms could be run in parallel, and even gBSS could be run with both the CPLEX and XPRESS solvers, since one might offer an advantage but this cannot be determined beforehand. Modern communications are reaching a point where such resources could be shared between sites within an enterprise, although their use would have to be managed.

Further developments to the system should include means to remove or highlight infeasible orders. Work on determining why a scheduling problem is infeasible would be
useful too.

A model and various routines which form the basis of an integrated decision support system have been described in previous chapters, and in this chapter a range of decision triggers have been proposed. The next step is therefore to try out the system with various triggers on some case studies, and this is described in the next chapter.

In addition the system was tested on a large number of smaller examples, a range of which are described briefly in Appendix C.
This chapter describes two case studies for integrated optimal scheduling and supervisory control. The first is the batch pilot plant at Imperial College. This provides a reasonably complicated plant, but remains well within the bounds of what can be scheduled in a reasonable length of time. It therefore provides a suitable test bed for extensive studies in various rescheduling strategies.

The second is the test plant for the BATCIME (BATch Control Integrated Manufacturing Execution) project. This was a European Community funded project with the objective of examining the interface between CIM systems and wider business systems, and as part of this project a model of a reference plant was put together, the aim being for the plant to contain just about all of the features and difficulties present within batch processing. This plant is based around the Imperial College batch pilot plant, but has a second plant in parallel, together with downstream separation processes, and an increased number of products and materials. This second case study therefore examines the bounds of what is possible with current technology and techniques.

9.1 The Batch Pilot Plant

The batch pilot plant at Imperial College was designed to contain as much complexity for its size as possible, and therefore although it may not contain a huge number of units, the complex routing and connectivity give great flexibility in its operation.

The plant consists of two feed tanks, a reactor, two product tanks, a CIP preparation vessel, and assorted feed and product storage (which does not actually appear as tanks in the real plant, since feed comes from dry storage and the mains water supply, and product is sent to the drain). The flowsheet is shown in Figure 9.1.

The processes are the degradation of various starches by an enzyme into various sugar solutions. The dry starch is mixed with water to form a paste. This is heated and then the enzyme is added, and the mixture held through a temperature profile while the starch
is broken down into a sugar. The degradation of two types of starch, and hence two products, is studied, with the operational constraint that units must be cleaned when changing between products. The STN for the process is shown in Figure 9.2.

![Figure 9.1: Flowsheet of the Pilot Plant.](image)

The standard mode of operation mixes the starch and water in a feed tank, T.1 or T.2. The resulting paste is then transferred to the reactor, T.3, where it is rapidly heated by steam sparging, (direct injection of steam). The enzyme is added and the temperature held using the reactor jacket. The product is discharged to a product tank, T.4 or T.5, where it is held while a sample is taken. The product is then transferred to a final product storage tank.

However an alternative process takes the paste made in a feed tank and transfers it directly to a product tank. There it is heated by recirculating the material through the external heat exchanger HE1, the enzyme is added, and the material held at the correct temperature profile again using HE1. This process takes longer than the standard process, but has the advantage that it avoids using the reactor, T.3, which is usually the bottleneck. A more detailed description of the process and the detailed control systems can be found in Liu (1995).

There is therefore a constantly changing tradeoff as to which is the best process to execute at any particular time. In order to add a changing situation over time to this flexible environment, a number of orders for the two products are placed, as shown in Table 9.1. The orders are spread over a few days, have various values, and are also placed at various times. Before an order is placed the system has no knowledge that it will be
placed, and therefore it can only plan how to fulfill that order after it has been placed. Some orders also have minimum amounts which must be delivered without fail.

![State Task Networks for the Pilot Plant Recipes.](image)

Figure 9.2: State Task Networks for the Pilot Plant Recipes.

Once the general model for this situation has been written, the integrated system described in Chapter 7 may be run for 65 hours, i.e. until after the last order in this set is due. The performance of the system is judged by the value obtained from these orders, subject to the cost associated with the waste from the cleaning process. (Cleaning a tank has a cost of 1.0).

The solution time allowed for the MILP solver used in these experiments was maintained so that a solution was returned within the inviolate period. It was found that for this particular problem the XPRESS solver gave results slightly more quickly than the CPLEX solver. The simulation model could be run at up to ten times real speed, but this then increases by a factor of ten the time in which $gBSS$ returns a solution (e.g. If $gBSS$ returns a solution after 5 minutes, 50 minutes of simulation will have occurred). Therefore to ensure consistent results, a reasonable time in which to reschedule, and a reasonable run time for each experiment, the model was always run at twice real speed.
9.1.1 Batch Pilot Plant Results

The first runs were conducted with various scheduling horizons. The only rescheduling criterion used was a fixed update period. For all of these results the schedule obtained from gBSS is always implemented on-line. The objective function criterion will not work in this case since a reduction in the objective function may sometimes be required when new orders with minimum quantities appear on the order slate. For practical reasons it is not feasible for a user to be present to confirm that the on-line schedule should be updated.

An arbitrary inviolate period of 15 minutes was chosen, although with the shorter horizons an optimal result was often returned in a much shorter period. The initial results for various schedule horizons and update frequencies are shown in Table 9.2.

<table>
<thead>
<tr>
<th>Product</th>
<th>Min:Max Amount</th>
<th>Value</th>
<th>Due after (hours)</th>
<th>Placed after (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cyc_dex</td>
<td>0:1000</td>
<td>1.0</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>cyc_dex</td>
<td>50:1000</td>
<td>2.0</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>cyc_dex</td>
<td>0:1000</td>
<td>1.0</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td>cyc_dex</td>
<td>50:1000</td>
<td>2.0</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>cyc_dex</td>
<td>0:1000</td>
<td>1.0</td>
<td>30</td>
<td>18</td>
</tr>
<tr>
<td>cyc_dex</td>
<td>50:1000</td>
<td>2.0</td>
<td>36</td>
<td>24</td>
</tr>
<tr>
<td>cyc_dex</td>
<td>0:1000</td>
<td>1.0</td>
<td>42</td>
<td>30</td>
</tr>
<tr>
<td>cyc_dex</td>
<td>50:1000</td>
<td>2.0</td>
<td>48</td>
<td>36</td>
</tr>
<tr>
<td>cyc_dex</td>
<td>0:1000</td>
<td>1.0</td>
<td>54</td>
<td>42</td>
</tr>
<tr>
<td>cyc_dex</td>
<td>50:1000</td>
<td>2.0</td>
<td>60</td>
<td>48</td>
</tr>
<tr>
<td>mal_dex</td>
<td>50:1000</td>
<td>2.0</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>mal_dex</td>
<td>0:1000</td>
<td>3.0</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>mal_dex</td>
<td>50:1000</td>
<td>1.0</td>
<td>21</td>
<td>9</td>
</tr>
<tr>
<td>mal_dex</td>
<td>0:1000</td>
<td>1.0</td>
<td>27</td>
<td>15</td>
</tr>
<tr>
<td>mal_dex</td>
<td>50:1000</td>
<td>2.0</td>
<td>33</td>
<td>21</td>
</tr>
<tr>
<td>mal_dex</td>
<td>0:1000</td>
<td>1.0</td>
<td>39</td>
<td>27</td>
</tr>
<tr>
<td>mal_dex</td>
<td>50:1000</td>
<td>1.0</td>
<td>45</td>
<td>33</td>
</tr>
<tr>
<td>mal_dex</td>
<td>0:1000</td>
<td>3.0</td>
<td>51</td>
<td>39</td>
</tr>
<tr>
<td>mal_dex</td>
<td>50:1000</td>
<td>2.0</td>
<td>57</td>
<td>45</td>
</tr>
<tr>
<td>mal_dex</td>
<td>0:1000</td>
<td>1.0</td>
<td>63</td>
<td>51</td>
</tr>
</tbody>
</table>

Table 9.1: Order slate for the case study.

The best results for each horizon used were obtained with frequent updates every 15 or 30 minutes. The best results overall came with short horizons of 6 or 8 hours. Therefore it would appear that the best way to operate this system is with frequent optimisation using a short horizon. This is possibly due to the fact that the short horizon leads to a small MILP which is more likely to be solved to optimality, or at least closer to it, than a larger problem. This is despite the fact that information about future orders is available 12 hours in advance.
### Table 9.2: Value added for various scheduling horizons and update intervals. (Backwards translation as described in Algorithm 5.1).

<table>
<thead>
<tr>
<th>Added Value</th>
<th>Scheduling Horizon</th>
<th>Update Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>10208.0</td>
<td>6 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>10694.0</td>
<td>6 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>9471.0</td>
<td>6 hours</td>
<td>45 minutes</td>
</tr>
<tr>
<td>7306.0</td>
<td>6 hours</td>
<td>60 minutes</td>
</tr>
<tr>
<td>10512.0</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>10314.0</td>
<td>8 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>10275.0</td>
<td>8 hours</td>
<td>45 minutes</td>
</tr>
<tr>
<td>9597.0</td>
<td>8 hours</td>
<td>60 minutes</td>
</tr>
<tr>
<td>10146.0</td>
<td>10 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>10742.0</td>
<td>10 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>10288.0</td>
<td>10 hours</td>
<td>45 minutes</td>
</tr>
<tr>
<td>9580.0</td>
<td>10 hours</td>
<td>60 minutes</td>
</tr>
<tr>
<td>9722.0</td>
<td>12 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>9591.0</td>
<td>12 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>9720.0</td>
<td>12 hours</td>
<td>45 minutes</td>
</tr>
<tr>
<td>9958.0</td>
<td>12 hours</td>
<td>60 minutes</td>
</tr>
<tr>
<td>10179.0</td>
<td>24 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>9935.0</td>
<td>24 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>8678.0</td>
<td>24 hours</td>
<td>45 minutes</td>
</tr>
<tr>
<td>8782.0</td>
<td>24 hours</td>
<td>60 minutes</td>
</tr>
</tbody>
</table>

### Table 9.3: Value added for various scheduling horizons and update intervals. (Forwards translation as described in Algorithm 5.2).

<table>
<thead>
<tr>
<th>Added Value</th>
<th>Scheduling Horizon</th>
<th>Update Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>12132.0</td>
<td>6 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>11323.0</td>
<td>6 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>11446.0</td>
<td>6 hours</td>
<td>45 minutes</td>
</tr>
<tr>
<td>10291.0</td>
<td>6 hours</td>
<td>60 minutes</td>
</tr>
<tr>
<td>12242.0</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>11899.0</td>
<td>8 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>11787.0</td>
<td>8 hours</td>
<td>45 minutes</td>
</tr>
<tr>
<td>9345.0</td>
<td>8 hours</td>
<td>60 minutes</td>
</tr>
<tr>
<td>12134.0</td>
<td>10 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>10399.0</td>
<td>10 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>9924.0</td>
<td>10 hours</td>
<td>45 minutes</td>
</tr>
<tr>
<td>9725.0</td>
<td>10 hours</td>
<td>60 minutes</td>
</tr>
<tr>
<td>9740.0</td>
<td>12 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>10051.0</td>
<td>12 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>10302.0</td>
<td>12 hours</td>
<td>45 minutes</td>
</tr>
<tr>
<td>11689.0</td>
<td>12 hours</td>
<td>60 minutes</td>
</tr>
<tr>
<td>10283.0</td>
<td>24 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>10186.0</td>
<td>24 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>10004.0</td>
<td>24 hours</td>
<td>45 minutes</td>
</tr>
<tr>
<td>10865.0</td>
<td>24 hours</td>
<td>60 minutes</td>
</tr>
</tbody>
</table>
It is interesting to note that if the common resource availability of fixed control recipes is not reduced to take account of fixed tasks at the beginning of the off-line scheduling horizon, as described in Section 6.3.2, then the added value for a scenario run was found to be typically 10% lower (with all the other parameters unchanged). This demonstrates the importance in this case of initialising the rescheduling accurately, which is unlikely to occur unless it is performed automatically.

The results in Table 9.2 used the backwards translation method outlined in Chapter 5. These situations were then repeated using the forwards translation method, and the results are shown in Table 9.3. It can be seen that in every case the forwards translation method gives better results.

Figures 9.3 and 9.4 show a reason why in this case the forwards method of translation often gives better results. In the gBSS schedule shown in Figure 9.3 the batches of product are run neatly in parallel through the reactor T3 and the product tanks T4 and T5. In particular, the cleaning of tank T5, shown with a red outline, occurs during a batch of product processed in the reactor, which is outlined in green. The backwards method of translation will schedule the batch of product after the cleaning batch, and this results in the schedule in SUPERBATCH shown in Figure 9.4. In this schedule the batch of product, again outlined in green is pushed back and does not start until after the cleaning batch, again shown in red has finished. This is because both these batches use heat exchanger HE3, and SUPERBATCH only allows the product batch to use this resource after the cleaning batch has finished. This results in a schedule that is far from the optimum calculated by gBSS.

On the other hand, the forwards translation method will result in an order of batches that puts the cleaning batch after the product batch, and the resulting SUPERBATCH schedule would retain the layout of the gBSS schedule as usual. The overall amount of product that can be produced over the 65 hour period is therefore often greater when the forwards translation method is used for this particular situation, and this method will be used in all further tests. A horizon of 8 hours and an update interval of 15 minutes also seemed to give good results, and will be used in future experiments.

The gPROMS model used in the first experiments had no variation in processing times from the nominal case. A series of runs was carried out with various percentages of variation in the model to examine how this degrades the value which is added over the order horizon. Each case was repeated seven times in order to get an average value for the added value. The effect of changing the horizon and update interval was then examined, as shown in Table 9.4. From these results it can be seen that as would be expected, as the variability in the model increases, so the value added falls. However this effect seems to be less pronounced when off-line rescheduling is carried out more frequently.
Figure 9.3: A gBSS Gantt chart for the pilot plant after 28 hours.

Figure 9.4: The resulting SUPERBATCH Gantt chart after translation.
Chapter 9. General Case Studies

<table>
<thead>
<tr>
<th>Variation (%)</th>
<th>Average Added Value</th>
<th>Scheduling Horizon</th>
<th>Update Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12147.4</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>0</td>
<td>10314.3</td>
<td>8 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>5</td>
<td>10995.2</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>5</td>
<td>10558.9</td>
<td>8 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>10</td>
<td>10712.7</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>10</td>
<td>10153.9</td>
<td>8 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>15</td>
<td>10280.2</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>15</td>
<td>9587.5</td>
<td>8 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>20</td>
<td>11471.1</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>20</td>
<td>10517.3</td>
<td>8 hours</td>
<td>30 minutes</td>
</tr>
</tbody>
</table>

Table 9.4: Value added for various percentages of variation in the model.

Another set of experiments examined the effect of changing the rounding coefficient used for calculating the task processing times from the step details. This value has the effect of making the scheduling model more conservative as its value increases. The results are shown in Table 9.5.

From these results it can be seen that there is no clear pattern, but that higher values for the rounding coefficient give marginally better results. The lack of any large differences in these results could be due to the small time discretisation used in the gBSS model, which at 15 minutes results in the model being fairly accurately based on the nominal times.

<table>
<thead>
<tr>
<th>Rounding</th>
<th>Added Value</th>
<th>Scheduling Horizon</th>
<th>Update Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>10735.0</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>0.25</td>
<td>11216.0</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>0.50</td>
<td>11005.0</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>0.75</td>
<td>12032.0</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
<tr>
<td>0.99</td>
<td>11570.0</td>
<td>8 hours</td>
<td>15 minutes</td>
</tr>
</tbody>
</table>

Table 9.5: Value added for various values of the rounding coefficient.

Another set of experiments looks at the effect of varying the inviolate period, which has so far been fixed at 15 minutes. The results are shown in Table 9.6. As the inviolate period increases, more of the schedule is fixed, and therefore the system is less flexible and the performance is degraded. However if the inviolate period is too short then some of the schedules returned by gBSS may not be implemented because batches which were assumed to be changeable have already started. Therefore there is a drop in performance with very short inviolate periods, and for this scenario the optimum appears to be around 10 minutes.

So far the only rescheduling criterion used has been a periodic update. The set of ex-
Table 9.6: Value added for various inviolate period lengths.

The experiments shown in Table 9.7 shows the effect of choosing various options. The simulation model was given a variability of 10% so that the system has some disturbance to deal with in addition to the changing order slate.

Although the results are not conclusive, possibly because the 15 minute update is already dealing with most of the disturbances, there appears to be a slight improvement as more criteria are added, and therefore rescheduling is prompted more often.

Table 9.7: Value added for various rescheduling policies. (10% variation).

The experiments carried out have therefore explored the various arbitrary parameters chosen in the integrated system.

9.1.2 Batch Pilot Plant Conclusions

Firstly this case study has shown how the processing environment described by Liu (1995) can now also include automatic implementation of off-line schedules, as well as automatic initialisation of the off-line scheduling system. With the exception of the detailed dynamics in the gPROMS model, the entire support structure of gPROMS, SUPERBATCH, ControlLink and gBSS has been created and linked into a single entity for much the same effort as would be required to create the SUPERBATCH model.

Secondly this has been a valuable test bed to examine and compare the performance achieved by the system for various rescheduling triggers and other system parameters.
Generally it is found that the value added improves as rescheduling is carried out more frequently. Beyond this it is hard to make generalisations for other plants, so experimentation with various parameters would be required for each case. However the development of an integrated system using the approach outlined in this thesis allows the experimental system to be created with minimal effort.

9.1.2.1 Dealing with major upsets

As an aside, if a recipe failed the sampling while waiting in a product tank, then so long as the SUPERBATCH system is able to change the material present in the product tank in its model and reflect this in the RUAPS information read by gBOSS++, the integrated system should be able to cope with this.

A recipe would be required that takes the failed state, e.g. MALDEX.FAILED which should have a negative price, and either partially or fully reworks it or blends it with another batch, and then the problem should be dealt with automatically. gBSS will be automatically initialised with the failed state in the product tank, and will then fit the batches required to remove the material into the next schedule generated. The standard gBSS problem will remain the same size since the rework tasks will only be considered if the failed state is present in the problem.

9.2 The BATCIME Reference Plant

The BATCIME project was a project of the European Batch Forum looking at the interaction between batch control systems and wider business functions. As part of this project a reference plant was developed. This plant seeks to include just about every conceivable constraint commonly present in batch processing.
Figure 9.5: Flowsheet of the BACIEME Plant.
The plant is based around two instances of the Imperial College batch pilot plant running in parallel, together with a downstream separation section and packing lines. The overall flowsheet is shown in Figure 9.5. There are therefore far more pieces of equipment than in the batch pilot plant case study, and these are listed in Table 9.2.

The process in each of the batch pilot plant sections is again the degradation of starch into sugar. However the resulting sugar solution is now passed to a crystalliser where most of the water is evaporated off. This leaves a sludge which has further water removed in a centrifuge, before passing through a dryer. The final crystalline powder is stored in
a silo before being packed into bags, which are kept in a final product warehouse.

![Condensed STN for the BATCIME Plant.](image)

Figure 9.7: Condensed STN for the BATCIME Plant.

Depending on the type of starch used, (wheat or corn starch), and which enzyme is added to perform the degradation, one of three different sugars can be made, namely dextrins, cyclodextrins and maltose. Ten different final products are made, each being bags with various proportions of the three types of sugar, as shown in Table 9.2.

If the plant is operated so that the intermediate tanks T4a, T4b, T5a and T5b are used to mix and split batches, then the problem becomes extremely complex, as shown by the STN in Figure 9.6. This is because each intermediate tank must be modelled as a separate resource with distinct tasks taking material to and from that tank, in order for
### Table 9.8: Resources and Units for the BATCIME reference plant.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Units</th>
<th>Size</th>
<th>Materials stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS_warehouse</td>
<td>WS_hse</td>
<td>100</td>
<td>w_starch</td>
</tr>
<tr>
<td>CS_warehouse</td>
<td>CS_hse</td>
<td>100</td>
<td>c_starch</td>
</tr>
<tr>
<td>Bulk_liq_1</td>
<td>ST_1</td>
<td>50</td>
<td>proc_water_1</td>
</tr>
<tr>
<td>Bulk_liq_2</td>
<td>ST_2</td>
<td>50</td>
<td>proc_water_2</td>
</tr>
<tr>
<td>Min_liq_1</td>
<td>ST_3</td>
<td>5</td>
<td>additive_A</td>
</tr>
<tr>
<td>Min_liq_2</td>
<td>ST_4</td>
<td>5</td>
<td>additive_B</td>
</tr>
<tr>
<td>Min_liq_3</td>
<td>ST_5</td>
<td>5</td>
<td>additive_C</td>
</tr>
<tr>
<td>Det_liquids</td>
<td>ST_6</td>
<td>3</td>
<td>detergent</td>
</tr>
<tr>
<td>Water_line</td>
<td>W_line</td>
<td>1000</td>
<td>water</td>
</tr>
<tr>
<td>Debagger</td>
<td>Debagger</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Feed_tanks</td>
<td>T1a, T2a, T1b, T2b</td>
<td>1, 1, 2, 2</td>
<td></td>
</tr>
<tr>
<td>Reactors</td>
<td>T3a, T3b</td>
<td>1, 2</td>
<td></td>
</tr>
<tr>
<td>Int_tanks</td>
<td>T4a, T4b, T5a, T5b</td>
<td>1, 2, 1, 2</td>
<td>dex_int, cyc_int, mal_int</td>
</tr>
<tr>
<td>Crylsrs</td>
<td>ICT25, ICT26</td>
<td>1, 2</td>
<td></td>
</tr>
<tr>
<td>Centrs</td>
<td>ICT23, ICT24</td>
<td>1, 2</td>
<td></td>
</tr>
<tr>
<td>Dryer</td>
<td>ICT22</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Silo_1</td>
<td>Silo_1</td>
<td>5</td>
<td>dry_dex</td>
</tr>
<tr>
<td>Silo_2</td>
<td>Silo_2</td>
<td>5</td>
<td>dry_cyc</td>
</tr>
<tr>
<td>Silo_3</td>
<td>Silo_3</td>
<td>5</td>
<td>dry_mal</td>
</tr>
<tr>
<td>Pack_lines</td>
<td>PL_1, PL_2</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>A_warehouse</td>
<td>A_hse</td>
<td>10</td>
<td>prod_A</td>
</tr>
<tr>
<td>B_warehouse</td>
<td>B_hse</td>
<td>10</td>
<td>prod_B</td>
</tr>
<tr>
<td>C_warehouse</td>
<td>C_hse</td>
<td>10</td>
<td>prod_C</td>
</tr>
<tr>
<td>D_warehouse</td>
<td>D_hse</td>
<td>10</td>
<td>prod_D</td>
</tr>
<tr>
<td>E_warehouse</td>
<td>E_hse</td>
<td>10</td>
<td>prod_E</td>
</tr>
<tr>
<td>F_warehouse</td>
<td>F_hse</td>
<td>10</td>
<td>prod_F</td>
</tr>
<tr>
<td>G_warehouse</td>
<td>G_hse</td>
<td>10</td>
<td>prod_G</td>
</tr>
<tr>
<td>H_warehouse</td>
<td>H_hse</td>
<td>10</td>
<td>prod_H</td>
</tr>
<tr>
<td>I_warehouse</td>
<td>I_hse</td>
<td>10</td>
<td>prod_I</td>
</tr>
<tr>
<td>J_warehouse</td>
<td>J_hse</td>
<td>10</td>
<td>prod_J</td>
</tr>
</tbody>
</table>
the use of the tanks to be fully modelled. This can be seen in the STN as separate paths for each possible route through the plant.

However if on the other hand batch integrity must be maintained until the silos, then the STN can be simplified to that shown in Figure 9.7. This is of a more reasonable size for gBSS, especially given that there is a cleaning requirement between different products, and therefore a number of unit conditions, which significantly increases the size of the MILP problem.

<table>
<thead>
<tr>
<th>Product</th>
<th>Percentage Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Maltose</td>
</tr>
<tr>
<td>A</td>
<td>75</td>
</tr>
<tr>
<td>B</td>
<td>75</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>50</td>
</tr>
<tr>
<td>E</td>
<td>25</td>
</tr>
<tr>
<td>F</td>
<td>50</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>100</td>
</tr>
<tr>
<td>J</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 9.9: Composition of the various products.

9.2.1 BATCIME Reference Plant Results

An initial order slate for three products, as shown in Table 9.10, was used as an objective for an initial schedule.

<table>
<thead>
<tr>
<th>Product</th>
<th>Minimum:Maximum Amount</th>
<th>Due after (hours)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>2.0:200.0</td>
<td>96</td>
<td>1.01</td>
</tr>
<tr>
<td>H</td>
<td>2.0:200.0</td>
<td>96</td>
<td>1.02</td>
</tr>
<tr>
<td>I</td>
<td>2.0:200.0</td>
<td>96</td>
<td>1.03</td>
</tr>
</tbody>
</table>

Table 9.10: Initial BATCIME order slate.

The resulting schedule from gBSS is shown in Figure 9.8. This is automatically translated without any problems to give the SUPERBATCH schedule shown in Figure 9.9.

One possible disruption would be the addition of another order after the initial schedule has been running for a couple of hours. An order for between 0.0 and 100.0 of product A, due after 96 hours with a value of 2.0 is added to the order slate, and a gBSS model is created reflecting the current position in the plant. The solution from gBSS is shown in
Figure 9.8: Initial gBSS Gantt chart for the BATCIME plant.

Figure 9.9: Initial SUPERBATCH Gantt chart for the BATCIME plant.
Figure 9.11. Again this is translated without trouble to give the SUPERBATCH schedule shown in Figure 9.12.

Figure 9.10 shows the common resource profile from gBSS for this result. The reduction in the availability of steam at the beginning due to the fixed recipe can be clearly seen.

An alternative disruption would be to take a unit out of action. Figure 9.13 shows the result from gBSS when unit T1A is taken out of service for 72 hours. In the initial schedule this unit is heavily utilised, but the production can remain more or less unchanged since T2a may be used as an alternative to T1a. It is the reactors T3a and T3b which are the bottleneck, and therefore even with T1a out of service the overall production remains the same. Again the gBSS schedule is successfully translated into SUPERBATCH input and gives the schedule shown in Figure 9.14.

9.2.2 BATCIME Reference Plant Conclusions

This case study has primarily shown that the techniques outlined in this thesis are quite able to cope with large and complex plants. The bottleneck in system performance remains the optimal scheduling, but this is likely to always be true, since this is by far the largest and most combinatorially complex problem. However the combination of these techniques for integration which allow for scheduling models to be for more aggregated in view than those used for supervisory control, together with incremental advances in computing power and MILP solver algorithms, has allowed the constraints on what size of plants and processes may be operated using these techniques to be relaxed significantly over those achieved by Chua (1995).
Figure 9.11: gBSS schedule for the BATCIME plant with an extra order.

Figure 9.12: SUPERBATCH schedule for the BATCIME plant with an extra order.
Figure 9.13: gBSS schedule for the BATCIME plant with a unit out of action.

Figure 9.14: SUPERBATCH schedule for the BATCIME plant with a unit out of action.
Air Products Case Study

This chapter describes a third case study, based upon the “Multi Purpose Reactor System”, (MPRS), at the Air Products and Chemicals Inc. site at Clayton in Manchester.

10.1 Plant and Process Description

The plant consists of four reactors, with two intermediate storage tanks. Most raw materials, intermediates and products are stored in drums, and pumped to and from the reactors through a number of drumming up and drumming off lines. The connectivity of these lines and tanks is shown in Figure 10.1.

Eleven final products are produced, and these are made from combinations of eight intermediates and thirty-three raw materials. The products are generally made in a single reactor, and the recipes consist of a number of additions interspersed with heating, cooling and mixing steps. Orders are given for a two week horizon as shown in Table 10.1, with the full list of materials shown in Table 10.5.
Chapter 10. Air Products Case Study

<table>
<thead>
<tr>
<th>Product</th>
<th>Quantity</th>
<th>Due after (hours)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT10</td>
<td>0</td>
<td>20</td>
<td>240</td>
</tr>
<tr>
<td>INT11</td>
<td>0</td>
<td>3</td>
<td>168</td>
</tr>
<tr>
<td>INT12</td>
<td>0</td>
<td>10</td>
<td>336</td>
</tr>
<tr>
<td>INT13</td>
<td>40</td>
<td>40</td>
<td>168</td>
</tr>
<tr>
<td>INT14</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INT15</td>
<td>0</td>
<td>30</td>
<td>336</td>
</tr>
<tr>
<td>INT16</td>
<td>0</td>
<td>60</td>
<td>192</td>
</tr>
<tr>
<td>INT17</td>
<td>43</td>
<td>43</td>
<td>120</td>
</tr>
<tr>
<td>INT18</td>
<td>15</td>
<td>15</td>
<td>96</td>
</tr>
<tr>
<td>INT19</td>
<td>0</td>
<td>13</td>
<td>336</td>
</tr>
<tr>
<td>INT20</td>
<td>0</td>
<td>20</td>
<td>336</td>
</tr>
</tbody>
</table>

Table 10.1: Order slate for APCI Case Study.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Size</th>
<th>Used for making (storing):</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>4.0</td>
<td>INT1, INT7, INT10, INT11, INT12, INT13</td>
</tr>
<tr>
<td>R2</td>
<td>9.0</td>
<td>INT8, INT15, INT16, INT17</td>
</tr>
<tr>
<td>R3</td>
<td>9.0</td>
<td>INT2, INT3, INT4, INT5, INT6, INT14, INT18</td>
</tr>
<tr>
<td>R4</td>
<td>14.0</td>
<td>INT19, INT20</td>
</tr>
<tr>
<td>ST1</td>
<td>24.0</td>
<td>(INT2)</td>
</tr>
<tr>
<td>ST2</td>
<td>18.0</td>
<td>(INT3)</td>
</tr>
</tbody>
</table>

Table 10.2: Units in the Case Study.

<table>
<thead>
<tr>
<th>INT1</th>
<th>INT7</th>
<th>INT10</th>
<th>INT11</th>
<th>INT12</th>
<th>INT13</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 10.3: Cleaning Matrix for Reactor 1.

<table>
<thead>
<tr>
<th>INT8</th>
<th>INT15</th>
<th>INT16</th>
<th>INT17</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 10.4: Cleaning Matrix for Reactor 2.
<table>
<thead>
<tr>
<th>Generic Material</th>
<th>Specific Material</th>
<th>State</th>
<th>Initial Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM1</td>
<td>RM1</td>
<td>RM1</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM2</td>
<td>RM2</td>
<td>RM2</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM3</td>
<td>RM3</td>
<td>RM3</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM4</td>
<td>RM4</td>
<td>RM4</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM5</td>
<td>RM5</td>
<td>RM5</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM6</td>
<td>RM6</td>
<td>RM6</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM7</td>
<td>RM7</td>
<td>RM7</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM8</td>
<td>RM8</td>
<td>RM8</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM9</td>
<td>RM9</td>
<td>RM9</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM10</td>
<td>RM10</td>
<td>RM10</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM11</td>
<td>RM11</td>
<td>RM11</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM12</td>
<td>RM12</td>
<td>RM12</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM13</td>
<td>RM13</td>
<td>RM13</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM14</td>
<td>RM14</td>
<td>RM14</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM15</td>
<td>RM15</td>
<td>RM15</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM16</td>
<td>RM16</td>
<td>RM16</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM17</td>
<td>RM17</td>
<td>RM17</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM18</td>
<td>RM18</td>
<td>RM18</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM19</td>
<td>RM19</td>
<td>RM19</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM20</td>
<td>RM20</td>
<td>RM20</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM21</td>
<td>RM21</td>
<td>RM21</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM22</td>
<td>RM22</td>
<td>RM22</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM23</td>
<td>RM23</td>
<td>RM23</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM24</td>
<td>RM24</td>
<td>RM24</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM25</td>
<td>RM25</td>
<td>RM25</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM26</td>
<td>RM26</td>
<td>RM26</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM27</td>
<td>RM27</td>
<td>RM27</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM28</td>
<td>RM28</td>
<td>RM28</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM29</td>
<td>RM29</td>
<td>RM29</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM30</td>
<td>RM30</td>
<td>RM30</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM31</td>
<td>RM31</td>
<td>RM31</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM32</td>
<td>RM32</td>
<td>RM32</td>
<td>9000.0</td>
</tr>
<tr>
<td>RM33</td>
<td>RM33</td>
<td>RM33</td>
<td>9000.0</td>
</tr>
<tr>
<td>INT1</td>
<td>INT1</td>
<td>INT1</td>
<td>3.0</td>
</tr>
<tr>
<td>INT2</td>
<td>INT2</td>
<td>INT2</td>
<td>8.0</td>
</tr>
<tr>
<td>INT3</td>
<td>INT3</td>
<td>INT3</td>
<td>3.0</td>
</tr>
<tr>
<td>INT4</td>
<td>INT4</td>
<td>INT4</td>
<td>0.0</td>
</tr>
<tr>
<td>INT5</td>
<td>INT5</td>
<td>INT5</td>
<td>0.0</td>
</tr>
<tr>
<td>INT6</td>
<td>INT6</td>
<td>INT6</td>
<td>0.0</td>
</tr>
<tr>
<td>INT7</td>
<td>INT7</td>
<td>INT7</td>
<td>0.0</td>
</tr>
<tr>
<td>INT8</td>
<td>INT8</td>
<td>INT8</td>
<td>0.0</td>
</tr>
<tr>
<td>INT10</td>
<td>INT10</td>
<td>INT10</td>
<td>0.0</td>
</tr>
<tr>
<td>INT11</td>
<td>INT11</td>
<td>INT11</td>
<td>0.0</td>
</tr>
<tr>
<td>INT12</td>
<td>INT12</td>
<td>INT12</td>
<td>0.0</td>
</tr>
<tr>
<td>INT13</td>
<td>INT13</td>
<td>INT13</td>
<td>0.0</td>
</tr>
<tr>
<td>INT14</td>
<td>INT14</td>
<td>INT14</td>
<td>0.0</td>
</tr>
<tr>
<td>INT15</td>
<td>INT15</td>
<td>INT15</td>
<td>0.0</td>
</tr>
<tr>
<td>INT16</td>
<td>INT16</td>
<td>INT16</td>
<td>0.0</td>
</tr>
<tr>
<td>INT17</td>
<td>INT17</td>
<td>INT17</td>
<td>0.0</td>
</tr>
<tr>
<td>INT18</td>
<td>INT18</td>
<td>INT18</td>
<td>0.0</td>
</tr>
<tr>
<td>INT19</td>
<td>INT19</td>
<td>INT19</td>
<td>0.0</td>
</tr>
<tr>
<td>INT20</td>
<td>INT20</td>
<td>INT20</td>
<td>0.0</td>
</tr>
<tr>
<td>Cleaner</td>
<td>Cleaner</td>
<td>Cleaner</td>
<td>9000.0</td>
</tr>
<tr>
<td>Waste</td>
<td>Waste</td>
<td>Waste</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 10.5: Materials in the Air Products Case Study.
Chapter 10. Air Products Case Study

### Table 10.6: Cleaning Matrix for Reactor 3.

<table>
<thead>
<tr>
<th></th>
<th>INT2</th>
<th>INT3</th>
<th>INT4</th>
<th>INT5</th>
<th>INT6</th>
<th>INT14</th>
<th>INT18</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT2</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>INT3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>INT4</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>INT5</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>INT6</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>INT14</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>INT18</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

### Table 10.7: Cleaning Matrix for Reactor 4.

<table>
<thead>
<tr>
<th></th>
<th>INT19</th>
<th>INT20</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT19</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>INT20</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

The equipment functionalities and sizes are shown in Table 10.1. When changing between most products the reactor must be cleaned as shown in Tables 10.1, 10.2 and 10.2. The cleaning batch takes twelve hours and has a cost of 1.2 units/batch.

### 10.2 Approach to the Case Study

The main feature which differentiates this plant and process from those previously studied is the use of the drumming lines, which are switched between batches during their execution.

![Figure 10.2: Scheduling two batches when the drumming line is not switched.](image)

In order to schedule this type of operation, SUPERBATCH must treat each step in the manufacture of each product as a separate recipe. Otherwise the priority for the use of a drumming line is given to a product as a whole, whereas the production rate may be significantly higher if the use of the line is switched between reactors during the manufacture of a batch of product. Figure 10.2 shows a simple example of this. The
second batch, (shown in blue), cannot begin until the first batch has finished with the drumming line. In Figure 10.3 it can be seen that the overall production rate is vastly improved if the priority on the drumming line is switched, even though the first batch is slightly delayed.

However if we treat each step as a separate task, and use a time discretisation which allows the use of the drumming lines to be examined, then there is no way that a schedule for two weeks of production can be found, since the problem will have in excess of 200,000 integer variables in the MILP formulation. If the horizon is reduced to a point where the problem may be feasibly attempted, then the minimization of cleaning changeovers and the tradeoffs between the variously valued orders cannot be made.

Therefore a twin level approach is proposed, as shown in Figure 10.4. At the top a 'coarse' representation of the problem is solved using gBSS. Here the drumming constraints are ignored, and therefore product recipes can be treated as a single task, a large time discretisation may be used, and a horizon of two weeks may be studied. This schedule can then be passed through a SUPERBATCH planner using a SUPERBATCH model with the drumming constraints removed. This effectively gives an upper bound on the problem, showing what could be achieved if the drumming line constraints could be scheduled so that there were no delays.

This coarse level schedule can then be used to set targets for a detailed problem. Each step is now modelled as a separate task in the detailed gBSS model, with the drumming line usage included and dummy states added to ensure the correct sequence of steps is processed, as shown in Figure 10.5. The time horizon can be quite short since the order of products has been fixed, and a small time discretisation allows the drumming line usage to be modelled accurately. The result of the detailed gBSS solution gives a sequence of steps which may be executed on-line in the SUPERBATCH monitor, and maximise the throughput through the drumming lines.

It could be argued that this plant is odd in that it should have been designed without the drumming line constraints. However the plant was initially designed to manufacture a
Figure 10.4: Overview of the approach to the Air Products Case Study.
Figure 10.5: Models used at each level for the Air Products Case Study.
single product which cascades through the three reactors, and the drumming lines are not a constraint. However changing market conditions led to a dramatic drop in the amount of this product required, and therefore the plant is now used for many products, in a role for which it was not designed. This highlights the need for flexible systems that can cope with situations and problems that are far from ideal.

10.3 Detailed Methodology

This section examines each of the data flows shown in Figure 10.4 and describes in more detail how the information is passed.

10.3.1 Model generation

The coarse \textit{gBSS} and \textit{SUPERBATCH} models are basically identical to what would be written normally, except that all common resources are ignored in the \textit{gBSS} model, and those resources are again not modelled in the coarse \textit{SUPERBATCH} model.

The detailed models are however quite different. The detailed \textit{gBSS} model only has process units; the storage resources are ignored. The State Task Network is made up of a task for every step in each recipe. Each task produces a unique output state material, which is an input state to the next step in the master recipe. These states may be stored in the reactor which produced them, but nowhere else, and thereby ensure that the processing steps are carried out in sequence. The \textit{gBSS} model therefore operates at a finer level of detail than most models for optimal scheduling which would normally aggregate a number of steps into a single task.

The detailed \textit{SUPERBATCH} model is closer to what would normally be written, except each step is given as a separate master recipe. The interface with the plant mimic is therefore unchanged, since at this level the information passed is in terms of unit operations and these remain unchanged.

10.3.2 Translation of the coarse \textit{gBSS} schedule

The translation of the coarse \textit{gBSS} output into a list of products to be executed in the coarse \textit{SUPERBATCH} model is identical to the standard translation outlined in Chapter 5.

10.3.3 Setting targets for detailed scheduling

Once the list of coarse control recipes has been passed through the \textit{SUPERBATCH} planner to give detailed timings for each step, targets for the detailed level scheduling can be found, as outlined in Algorithm 10.1.
Chapter 10. Air Products Case Study

Algorithm 10.1 (Setting targets for detailed scheduling).

For each fixed coarse control recipe...
1 For the unit performing this task ... 
   1.1 Find first step of this recipe finishing before now + LOOKAHEAD 
   1.2 mr_name.step_name.OS gets an order from start of step to end of the horizon.
2 For each future control recipe ... 
   2.1 If it starts before now + 8 hours... 
      2.1.1 For the unit performing this task ... 
         2.1.1.1 Find first step of this recipe finishing before now + LOOKAHEAD 
         2.1.1.2 mr_name.step_name.OS gets an order from start of step to end of the horizon.

The list of coarse control recipes is divided into those that are fixed, because they have start times in the coarse SUPERBATCH schedule before the beginning of the detailed scheduling horizon, and those that are still in the future. For each of the fixed coarse control recipes each process unit is examined. The step in this control recipe, which finishes before the lookahead period into the coarse schedule, is found. The dummy output state from this step is then set as an order at the time at which this step finishes.

10.3.4 Translation of detailed gBSS schedule

The task of taking the detailed gBSS schedule of steps and putting them into SUPERBATCH as a list of recipes to be executed should be fairly simple since there is a one-to-one mapping between the tasks in the gBSS model and the recipes in the SUPERBATCH model.

However, because the treatment of materials has been changed to enforce the order of the steps, and the real materials ignored, several checks must be made to ensure that the schedule is feasible.

Firstly, where an intermediate is made in one reactor to be consumed almost immediately in a parallel reactor, this linkage no longer appears in the detailed gBSS model, so we must check with a mass balance that the material is available.

Secondly, if there are two targets for the same product, say one order for the output state of step 6, the final step of a product, and another for the output state of step 3, then it is possible for a feasible gBSS solution where the orders are the wrong way round, as shown in Figure 10.6. The required solution, shown first, is for the output state of step 6 to be delivered first, then the output state of step 3. However, despite the fact that the delivery from step 6 is greatly delayed and has a higher priority, it is possible for the output from step 3 to be delivered first and then an entire batch made afterwards.
to fulfil the first order. This solution is severely suboptimal, but gBSS was found to sometimes return results such as this, especially when the solution was stopped because it had reached its time limit.

The overall result is that the sequence of steps obtained from gBSS is not feasible. Another possible reason for an infeasible sequence of steps is that the mass balance may have disrupted the order, or even removed a step from the schedule, since it is carried out on individual steps, rather than the whole recipe.

To overcome these infeasible sequences, the list of tasks can be analysed to ensure that the steps of each product always run in sequential order, except when step 1 follows the final step in a recipe.

These measures result in ensuring that the schedules can be executed on-line in SUPERBATCH.

10.3.5 Initialising the detailed gBSS model

The detailed gBSS model is initialised from the ruaps information from the on-line SUPERBATCH monitor, following the standard techniques described in Chapter 6.

10.3.6 Removing completed products from the coarse schedule

Once a product recipe has been completed in actual operation, it can be removed from the list of recipes that are providing targets for the detailed level. Therefore for each recipe at the coarse level that should have finished according to the coarse SUPERBATCH schedule, the steps that have been completed are checked, and if all the steps in the recipe have been completed then both the coarse recipe and the detailed steps are removed from the lists.
10.3.7 Initialising the coarse gBSS model

The coarse gBSS model is initialised from the ruaps information from the coarse SUPERBATCH planner, following the standard techniques described in Chapter 6. All recipes that have been used to set targets at the detailed level are fixed, since once started these recipes must be completed.

10.3.8 System Parameters

The four level system introduces a relatively large number of arbitrary parameters. These are discussed in turn, highlighting their effect on the performance of the integrated system.

10.3.8.1 Inviolate period and gBSS solution time

The inviolate period is the length of time into the future at which the plant status is taken, since this will hopefully approximate the status of the plant when, (or soon after), scheduling has finished. This fixes control recipes in the schedule which will have already started by the time off-line scheduling is completed.

Therefore this quantity is linked with the solution time limit given to gBSS, since this will stop the scheduling process when the inviolate period is about to be exceeded.

The longer the solution time the greater the chance that a solution will be found or the better the quality of the solution in some cases. However as the inviolate period grows the schedule becomes more fixed and cannot be adapted in the short term in reaction to changes from nominal event timings.

10.3.8.2 Time horizon of detailed gBSS model

The time horizon over which gBSS generates detailed schedules must be at least long enough for feasible production of the targets set, otherwise no solution will be found. However a longer horizon leads to longer solution times.

As actual production falls behind what is predicted in SUPERBATCH at the coarse level without the drumming constraints, the number of steps to be scheduled at the detailed level grows. The number of orders in the detailed problem is limited to two on each reactor, and so the amount of production can never exceed two complete batches on each reactor, and this can be used as a guide to the worst case.

The coarse level SUPERBATCH schedule is brought back into line when rescheduling at the coarse level is completed, and therefore the frequency of this can be used to keep the two levels reasonably matched. This keeps the horizon of the detailed model at a reasonable level, since the targets from the coarse level will be less demanding. If the
two levels are not kept together, the coarse model runs ahead and the targets become too optimistic.

10.3.8.3 The update time of the detailed loop

The period at which the detailed loop updates is set to ensure that there are sufficient steps actually scheduled on the plant so that the plant is never idle due to it having run out of things to do.

Since only one scheduling problem is solved in gBSS at a time, if the update is set below the time allowed for a solution, the scheduling may well often become continuous.

10.3.8.4 The update time of the coarse loop

Since the coarse loop simply sets the ordered list of products that will be made, this loop is updated on a less frequent basis. The update on this loop however takes priority over the detailed loop since without a coarse plan a detailed plan cannot be made.

The coarse loop should be updated whenever there exists a significant difference between the planned production in the coarse SUPERBATCH model and what has actually taken place.

10.3.8.5 The lookahead period into the coarse schedule

The longer this period then the more has to be scheduled at the detailed level, and therefore solution times will increase. However a better overall schedule is likely to result as the time increases.

10.3.8.6 The number of detailed orders on each unit

This number limits how much production is scheduled on each unit. This prevents overloading and an infeasible schedule as too much is required of one unit for the time horizon given at the detailed level.

A default value of two is used, since this allows the remainder of the current batch, as well as the whole of the next batch to be considered. Beyond that is probably too far into the future to be fixed, since by the time the batch after the current one, actual operation may be significantly different from the nominal case.

It does however shift the optimisation problem at the detailed level, since one reactor may continually be allowed to slip behind for the good of the whole. Therefore priorities could be given to either units or batches.
10.4 Case Study Results

The system is initialised with the given initial amounts of raw materials and intermediates at the beginning of the two week horizon. The equipment is idle and there are no activities scheduled. The coarse loop has precedence over the detailed loop since without a result at the coarse level there is nothing to schedule at the detailed level. Therefore the coarse gBSS level is initiated and the result read and passed through SUPERBATCH. Then the detailed gBSS level takes the first set of targets from the coarse SUPERBATCH schedule and the results are passed to the on-line SUPERBATCH monitor as an ordered list of steps to be executed. The monitor then manages the execution of these steps in the gPRO/MS model simulating the plant control system. The simulation is run on a separate machine since it is a constant process, but the other applications are run on a single machine. Performance might be improved if the load was shared, but the system is also more at risk if a machine becomes unavailable.

![Gantt chart](image)

Figure 10.7: Example gBSS Gantt chart from the coarse model.

After specified periods the coarse and detailed gBSS models are recreated given the current plant position and executed either to update the coarse SUPERBATCH schedule so that the targets remain achievable, or to update the list of steps to be executed on-line so that this remains optimal and up to date.
Figure 10.8: Example SUPERBATCH Gantt chart from the coarse model.
Figure 10.7 shows a typical result from the coarse gBSS model while the system is running. It can be seen that most of the tasks have a duration of one time step, (twelve hours), with the rest being of two or three times this. When such a schedule is passed through the SUPERBATCH planner, a schedule such as that shown in Figure 10.8 is obtained. Here the exact timings for each step in each product are used.

Figure 10.9 shows a typical result from the detailed gBSS model. The very short tasks are the waiting times inserted during the execution of a recipe while a drumming resource is unavailable. It can be seen that the tasks are shifted as far to the left as possible due to the penalties for lateness in the objective function.

When these new steps, (along with the previous ones), are passed through the SUPERBATCH planner, a schedule such as that shown in Figure 10.11 is obtained. This is similar to the coarse level schedule shown in Figure 10.8, except that the drumming constraints have been included, and there is more waiting time in the schedule. However the detailed gBSS level has ordered the priority on the steps so that this waiting time is minimised.

The sequence of products generated by the coarse level was roughly what would have been generated manually. However the cost of cleaning had to be increased from that given initially, since the solution otherwise was clearly suboptimal. When run for over 24 CPU hours the same initial coarse problem found no improvements over the schedule found within 30 CPU minutes.

It was found that using the non-zero lower bounds on the required amounts of material to be produced sometimes led to no feasible schedule being produced at the coarse level. Therefore the lower bound was relaxed to zero for each of the products, and the lower bound was in any event achieved for each of the products. This is due to the large time discretisation used at the coarse level, since if a product takes 7 hours to produce, the coarse model shows that only 2 batches can be made in a 24 hour period, when in fact 3 would be possible. This sort of problem is often corrected when rescheduling is repeated a few hours later.

Eventually the two weeks of integrated operation are simulated and the amount of each product that has been produced in time for the due date can be found from the on-line schedule. In addition the percentage of time when each reactor was actually processing can be found. These results can be used to compare the effectiveness of the system under various conditions and with actual practice.

1Using a Sun SPARC Ultra 2
### Figure 10.9: Example gBSS Gantt chart from the detailed model.

#### Key to Tasks

```
1. M_113_S_1
2. M_113_S_2
3. M_113_S_3
4. M_113_S_4
5. M_116_S_1
6. M_116_S_2
7. M_116_S_3
8. M_116_S_4
9. M_116_S_5
10. M_118_S_1
11. M_118_S_2
12. M_118_S_3
13. M_118_S_4
14. M_118_S_5
15. M_118_S_6
16. M_118_S_7
17. M_118_S_8
18. M_118_S_9
19. M_118_S_10
20. BO_R1_S_1
21. BO_R1_S_2
22. BO_R1_S_3
23. STM_113_S_1
24. STM_113_S_2
25. STM_113_S_3
26. STM_113_S_4
27. STM_116_S_1
28. STM_116_S_2
29. STM_116_S_3
30. STM_116_S_4
31. STM_116_S_5
32. STM_116_S_6
33. STM_118_S_1
34. STM_118_S_2
35. STM_118_S_3
36. STM_118_S_4
37. STM_118_S_5
38. STM_118_S_6
39. STM_118_S_7
40. STM_118_S_8
```

### Figure 10.10: List of tasks from the detailed model.
Figure 10.11: Example SUPERBATCH Gantt chart from the detailed model.
The first variable to be analysed is the frequency at which the detailed and coarse schedules are updated. The results are shown in Table 10.8. This shows the amount of each product made, the percentage utilisation of each reactor and the overall economic potential from the two week period based on the values of the orders fulfilled and the cost of cleaning performed.

<table>
<thead>
<tr>
<th>Detailed update frequency</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse update frequency</td>
<td>24</td>
<td>24</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Production of each product</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I10</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>I11</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>I12</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I13</td>
<td>40</td>
<td>40</td>
<td>36</td>
<td>40</td>
</tr>
<tr>
<td>I14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I15</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>18</td>
</tr>
<tr>
<td>I16</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>I17</td>
<td>45</td>
<td>45</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>I18</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>I19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I20</td>
<td>14</td>
<td>14</td>
<td>28</td>
<td>14</td>
</tr>
<tr>
<td>Utilisation of each reactor (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>81.9</td>
<td>76.2</td>
<td>83.40</td>
<td>84.07</td>
</tr>
<tr>
<td>R2</td>
<td>70.8</td>
<td>74.2</td>
<td>81.62</td>
<td>77.64</td>
</tr>
<tr>
<td>R3</td>
<td>85.1</td>
<td>72.9</td>
<td>80.85</td>
<td>82.32</td>
</tr>
<tr>
<td>R4</td>
<td>26.3</td>
<td>19.7</td>
<td>27.14</td>
<td>18.22</td>
</tr>
<tr>
<td>Net Economic total</td>
<td>162.619</td>
<td>174.098</td>
<td>178.036</td>
<td>202.219</td>
</tr>
</tbody>
</table>

Table 10.8: APCI results when loop update timings are changed.

It can be seen from the gBSS schedule, shown in Figure 10.7, that the utilisation for reactor R4 will be fairly low, since only two products are made there, and the orders for these products are not huge.

From the results it can be seen that updating the detailed schedule every hour and the coarse schedule every twelve hours gave the best overall performance. If the coarse schedule is updated too frequently then the solution of this problem can monopolise the processor, delaying the detailed scheduling. However if the coarse level is not updated fairly frequently then the targets for the detailed scheduling become too optimistic, since the real plant lags behind this upper bound due to the drumming constraints. Regular updates at the coarse level bring the two layers back together, and doing this every twelve hours appeared to be the best frequency.

A regular update of the detailed level ensures that the list of steps to be executed remains up to date, and ensures that the sequence of steps being processed is optimised taking into account all the steps which will be processed in the near future.
The coarse level gBSS model has a very large time discretisation interval of twelve hours, and therefore it might be expected that the way in which the task durations are rounded has a large impact on the performance of the system. Table 10.9 shows how the results change as the rounding coefficient is varied.

As the rounding coefficient increases, so the model becomes more conservative, and it is found that the system performs less well. Much of the decrease in performance is attributable to the fact that less of products I19 and I20 are made. These products are notable in that intermediates must be made in advance. The conservative scheduling model cannot find time to make the intermediates required early in the two week period, and cannot make up for this later on when it has been found that the other products have been made more quickly than expected, and therefore overall performance is degraded.

<table>
<thead>
<tr>
<th>Rounding</th>
<th>0.0</th>
<th>0.5</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detailed update frequency</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Coarse update frequency</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Production of each product</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I10</td>
<td>16</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>I11</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>I12</td>
<td>8</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>I13</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>I14</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I15</td>
<td>27</td>
<td>18</td>
<td>27</td>
</tr>
<tr>
<td>I16</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>I17</td>
<td>45</td>
<td>45</td>
<td>36</td>
</tr>
<tr>
<td>I18</td>
<td>25.5</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>I19</td>
<td>28</td>
<td>14</td>
<td>28</td>
</tr>
<tr>
<td>I20</td>
<td>28</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Utilisation of each reactor (%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>83.8</td>
<td>82.6</td>
<td>76.9</td>
</tr>
<tr>
<td>R2</td>
<td>70.0</td>
<td>74.6</td>
<td>73.6</td>
</tr>
<tr>
<td>R3</td>
<td>85.2</td>
<td>83.3</td>
<td>80.3</td>
</tr>
<tr>
<td>R4</td>
<td>54.6</td>
<td>29.4</td>
<td>30.3</td>
</tr>
<tr>
<td>Net Economic total</td>
<td>209.046</td>
<td>184.416</td>
<td>166.254</td>
</tr>
</tbody>
</table>

Table 10.9: APCI results when rounding is changed.

As seen before in Chapter 9, the underlying simulation model can introduce various degrees of uncertainty into the processing time for each step. The results from two week periods with various degrees of uncertainty are shown in Table 10.10.

As would be expected the overall performance degrades as the variability increases. However the fact that the reactor utilisation remains fairly constant, rather than dropping off with increasing variability, implies that the system is successfully adapting the sequence of steps to work around the variability. The fact that a level of variability in the simulation of 10% resulted in the best performance achieved in any run is probably due to good
fortune in the random way in which the processing times varied. i.e. the critical steps were finished ahead of time, while other steps were delayed when the unit would otherwise have been waiting.

### 10.5 Case Study Conclusions

Firstly this case study has shown the adaptability of a model-based approach to integration. The original prototype system has been modified to form a far more complicated system, which would have been extremely difficult to integrate otherwise. The system remains one where new recipes can be added with ease.

It has also been seen that although the system is fairly consistent when the simulation model introduces variability into the system, some of the parameters defined by the user, such as the rounding coefficient, can have a large impact on the overall system. The impact of given parameters cannot be determined before hand, and therefore experimentation would appear to be the only alternative. Since the system can generate its own simulation automatically however, the effort in carrying out such experiments is minimised.
In this thesis, a completely integrated manufacturing execution system which takes targets for plant production and then schedules and executes the required production in the plant has been described. This system is able to handle disruptions to the production either by modifying the schedule timings on-line or by completely reoptimising the schedule off-line. The entire system runs from a single plant model, ensuring model consistency between the various system parts, and vastly reducing the effort required to create and maintain such a system.

The key features of this work are therefore:

- Seamless information flow is possible from short-term scheduling to the plant and back again.
- The system is entirely model based and therefore flexible and adaptable to a very wide range of plants. It is also able to handle a range of software packages for the individual applications.
- The approach allows the scheduling model to be of an appropriate level of detail next to a more detailed supervisory model.
- The integrated system can be set up and tested very quickly.

The aim of integrating short-term off-line scheduling and supervisory control to allow free closed loop information transfer has therefore been achieved. The approach overcomes many of the difficulties found by Chua (1995), notably by allowing more aggregated scheduling models many of the difficulties with lengthy solution times are eased.

The model on which the system is based is able to contain any of the features listed in Table 3.2. It is therefore applicable to a very wide range of processing situations, and the algorithms for translating data have been tested on numerous combinations of these
Chapter 11. Conclusions

features. The system therefore does not degrade any of the flexibility in the underlying scheduling and supervisory control applications.

It should also be noted that the translation algorithms described in this thesis have been implemented and when run on large examples, (e.g. Chapter 9), the execution time has been found to be always near negligible, (i.e. well under a second), and this is without any special attention paid to optimising the implementation for speed. These algorithms are therefore practical on just about any size of problem.

11.1 Unresolved Issues

Although the integrated system has been shown to fully close the loop between short-term scheduling and supervisory control, there are a number of issues which are not fully finalised.

Firstly like many scheduling packages gBSS has a fixed horizon, and this can sometimes have a sizeable effect on the solutions returned. For example if the horizon is 8 hours, and one task takes 2 hours and another 3 hours. It may be obvious from the prices of the two products that the optimal operation is to carry out the 3 hour task repeatedly, whereas the optimal solution returned from the scheduling package will be two long tasks and a short task, since this maximises the objective function over the horizon. Worse still there is no reason why the shorter task is not started first, and therefore cannot be removed by rescheduling later. For the batch pilot plant case study it was seen that there was an ‘optimum’ horizon length. Whether this can only be found by trial and error, or whether a ‘natural’ horizon length can be calculated for a given plant and process is an open question.

Another related problem is what to do with product orders that lie beyond the current horizon, but require production to be started now if they are to be fulfilled. For the present they are turned into ‘soft’ orders at the end of the current horizon, but this may not be optimal. The natural place for this problem to be solved is at the higher planning level, since this is where the longer horizon is used, and therefore care should be taken that the results from such a system are “in tune” with the short-term scheduling function. Scheduling techniques based on “rolling horizons”, such as those described by Dimitriadis et al. (1997), also promise improvements in this area. Such techniques need not solve the entire horizon in detail, but only for the next day or so, leaving the rest of the horizon as an aggregate problem. This aggregate part of the horizon can be solved in detail later when further rescheduling is carried out, while the solution time for the information that is required immediately is reduced.

The current integrated system simply passes an ordered list of control recipes to the supervisory system, and the information about task timings, generated at the short-term
scheduling level, is discarded. In some situations, for example where tasks with heavy electrical power requirements are scheduled to coincide with periods of cheaper electricity during the night, some fixing of the timings in the supervisory schedule might be necessary.

Finally alternatives to using a maximal STN, (mSTN, Crooks, 1992), whenever there are multiple storage tanks to choose between, should be developed. Where the use of the storage is straightforward, the selection of the storage units might be better handled by heuristics either within the integrated system or in the supervisory system. This would greatly increase the size of plants with multiple storage tanks that can be scheduled within the integrated system. However, where the use of the storage tanks is integral to the scheduling of the plant there remains little alternative to the mSTN. Because of the way in which gBSS handles storage, only one storage unit can be in each storage resource, since gBSS cannot provide information on which storage is used.

11.2 Future Directions

The overall structure of the prototype system described in this thesis has been shown to work from a theoretical point of view. However in order to be used in an industrial setting, a number of additional features would be required.

The prototype system has been developed with few internal consistency checks in order to speed development. Any errors or inconsistencies in the sub-models are found when these are run in SUPERBATCH or gBSS. However such checks and diagnostics ought to be carried out during the initial data entry and processing.

Another requirement of an industrial system would be extensions to allow a “warm-start” where a plant is already running and the system automatically picks up what is taking place, rather than always starting with a blank initial schedule in the SUPERBATCH monitor.

The data entered is effectively static, in that unit sizes etc. cannot be changed. The ability to do this, together with a completely interactive user friendly graphical user interface for data entry and on-line updating would be required in a fully developed system. In addition many of the manual functions, such as updating the model or changing the on-line schedule would require authority checking and logging.

The updating of on-line schedules would probably also require some sort of supervisor authority in any real situation.

The actual position of the integrated system is also questionable. At one level a single unified package incorporating gBSS and SUPERBATCH could be envisaged. Then again the extensions to the model and input language could be incorporated within either gBSS or SUPERBATCH. Alternatively, the system could be completely stand-alone, with open
interfaces to conceivably any scheduling or execution package. This last alternative would be greatly strengthened by international standardisation in the interfaces between these types of packages. In the end the question of how such a system is developed is probably more a marketing issue than a technical one.

On the theoretical front, the main challenges lie in furthering the integration up the hierarchy into the enterprise-wide planning systems. An initial format for a file to pass orders into the current system has been described, but this requires testing with case studies, as well as research on what information needs to be passed up to the planning system, and when.

In addition to integration with operations systems, interfaces with other systems, such as maintainance, stock control and quality analysis systems must be examined.

There is also the possibility of studying the development of models during the design of batch plants and then their reuse during operation, so that models and associated knowledge is kept through these processes, leading to greater consistency and a reduction in the time spent creating models, creating more time for studying alternatives.

11.2.1 Future directions in scheduling

While not developing techniques in scheduling, this work is closely tied to this field. Work in scheduling already has the minimisation of solution time as a primary objective, but the following areas also present themselves as possibilities for improvement.

11.2.1.1 Algorithm selection

Scheduling algorithms tend to be very much horses for courses, rather than one size fits all, type solutions. Therefore techniques which help one to select an algorithm that is particularly suitable for a particular problem would be useful. As a first step, a common interface to a number of algorithms, (for example the original formulation of Kondili et al. (1993), the rolling horizon approaches of Dimitriadis et al. (1997), and the rescheduling approaches of Schilling (1997)), would simplify the task of finding a suitable algorithm, since a user would then only have to build a single model.

11.2.1.2 Schedule alternatives

Most scheduling tools find a single solution, whereas an integrated system would be able to cope with a stream of alternative schedules, allowing the analysis of each of them in a detailed model with perhaps alternative performance measures.
11.2.1.3 Schedule analysis

The development of techniques for deciding why one schedule is better than another would complement an integrated system. Scheduling results are often described as being better because of an improved objective function. However an operations manager may find it more useful to be told the reasons why a schedule is thought of as an improvement, for example which orders are more fully met. This might be particularly useful where a number of alternatives are presented.

Where no result is obtained because the problem is infeasible, it would be useful for the violating constraint to be identified, whereas at present the MILP is simply reported to have no solution. A common standard for reporting such issues however is likely to be a very distant prospect.


References


References


References


This appendix gives a brief description of the gBSS input language, paying particular attention to those features used by gBOSS++. A full description can be found in the user manual (gBSS User Manual, 1997).

The gBSS input consists of three parts, a STN description, a unit description and a problem statement. The STN is independent of any plant or problem, and the unit description is independent of the problem $^1$.

These three sections are held in three files, which are now described in turn.

A.1 State Task Network description

Firstly the states are listed, and if they are STABLE, i.e. they can be stored in the unit which produced them, this tag is added.

Then the tasks are listed. Each task may have an overall processing time, a list of input states and a list of output states. Each input state and output state has a proportion and a time at which it enters or leaves the task. The profile of any common resource usage over the task duration is also described.

![Example of a simple STN](image)

Figure A.1: Example of a simple STN.

$^1$An exception to this is that the unit description does contain details of common resource availability, whereas restrictions on unit availability appear in the problem statement.
Table A.1: Definition language for states and tasks in *gBSS*.

For the very simple STN shown in Figure A.1, the file shown in Table A.1 would be written.

A continuous task is described as shown in Table A.2. Each input state and output state has a fraction which is consumed or produced when the task begins, and a fraction which is consumed or produced when the task ends.

Table A.2: Definition language for continuous tasks in *gBSS*.

A.2 Unit description

The unit description file lists all the units, both those used for processing and those used for storage. An example is shown in Table A.3.

Processing units, such as the reactor, are given a capacity and a list of tasks which may be performed. Optionally the size of each task performed may be restricted by a UTILISATION. In this case the task size for *Make_C* in the reactor must be between 100 and 200.

The packing line is an example of a continuous unit, for which a RATE rather than a CAPACITY is defined. The tasks performed are again listed, and optionally a minimum run length is specified as the minimum run length is specified as the minimum length of time for which this unit must operate continuously if it is started.

Storage units are given a capacity, together with a list of states which may be stored. Units may have both a list of tasks performed and a list of states stored. In addition common resources and their availability are listed.
UNIT reactor
    CAPACITY 200.0
    PERFORM Make_C UTILISATION 0.5:1.0

UNIT packing_line
    RATE 4.0
    PERFORM packing
    MIN_RUN_LENGTH 1.0

UNIT storage_tank
    CAPACITY 450.0
    STORE C

UTILITY steam
    AVAILABLE 1.0 FROM 0.0 TO 2.0
    AVAILABLE 2.0 FROM 2.0 TO 4.0

Table A.3: Definition language for equipment in gBSS.

A.3 Problem description

The problem description statement is shown in Table A.4. A title is specified which can be used to distinguish this problem statement from others using the same basic model, and this is used in the output.

TITLE making_C_today

PROBLEM_TYPE SHORT_TERM_SCHEDULING

TIME_MODE REAL
INTERVAL 12.0

RECIPE_FILE factoryc
RESOURCE_FILE factoryc

METHOD
    MOC FOREIGN.MIP
    SOLVER XPRESS

STATE C
    INITIAL 200.0 IN storage_tank
    PRICE 1.34
    DELIVER 40.0:50.0 AT 12:00023/07/2000 VALUE 1.002
    DELIVER 25.0 FROM 12:00023/06/2000 TO 12:00001/07/2000 PRIORITY 2 SQUARE

Table A.4: Definition language for the problem statement in gBSS.

The problem type is specified as SHORT_TERM_SCHEDULING, since gBSS can also be used for campaign planning and design problems, which are not discussed here. The time mode can be REAL, where times are given in HH:MM/DD/MM/YYYY format, or RELATIVE...
where all times relate to 0 on the horizon.

The horizon, interval size, location of the other model files and the mode and solver to be used are also listed. The FOREIGN.MIP mode is specified when using the XPRESS or CPLEX solvers, which perform best for just about all problems.

Finally the initial information for each state is provided. An initial amount for each state, together with a location may be specified, and a price may be given, which will be used in the calculation of the objective function. Then a list of all the orders for that state are given. Orders may be given for a single amount of material or for a bounded range. The timing may be given as a single due date or again as a range. Where a range of time is given, a priority may be given, which means a penalty term is added to the objective function as the delivery is made further from the lower bound of the range. The term may increase either linearly or with the square of the delay.
This appendix gives a brief description of the SUPERBATCH input language, paying particular attention to those features used by gBOSS++. A full description can be found in the user manual (Superbatch User Manual, 1996).

The description may be broken down into five parts, each of which is now detailed in turn.

B.1 Plant description

The units in a plant are collected together as resources, which are sets of functionally equivalent units. The resources and units are described as shown in Table B.1.

```
RESOURCE feed_storage FEED
   UNIT A.store

RESOURCE reactors PROCESS
   UNIT reactor_1 MAX-WEIGHT 20.0

RESOURCE reactor_valves PROCESS ONLY PHASE-EXCLUSIVE
   UNIT V23 MAX-WEIGHT 1.0

RESOURCE intermediate_storage STORAGE
   UNIT B.store MAX-WEIGHT 200.0

RESOURCE product_storage PRODUCT
   UNIT C.store
```

Table B.1: Definition language for equipment in SUPERBATCH.

A resource may be of PROCESS type, in which case batch integrity is maintained, or storage, in which case material can be mixed between batches. FEED resources are storage resources which cannot be emptied since they are assumed to have a source from outside what is modelled, and PRODUCT resources are storage resources which cannot be filled since they are assumed to have a sink to outside what is modelled.
Resources may be described as PHASE-EXCLUSIVE, in which case they are locked when in use by a phase and cannot be used for any other process. Alternatively they may be BATCH-EXCLUSIVE, in which case they are locked for the whole duration of the batch which uses them.

<table>
<thead>
<tr>
<th>COMMON RESOURCE steam</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX-CAPACITY 4.0</td>
</tr>
</tbody>
</table>

Table B.2: Definition language for common resources in SUPERBATCH.

Common resources are simply defined as shown in Table B.2.

<table>
<thead>
<tr>
<th>OUTPUT OF RESOURCE reactor.valves IS INPUT TO reactors FOR UNITS (V23 TO reactor.1) (V24 TO reactor.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT OF RESOURCE reactors IS INPUT TO prod.storage FOR ALL UNITS</td>
</tr>
</tbody>
</table>

Table B.3: Definition language for flowsheets in SUPERBATCH.

A flowsheet detailing the connectivity between the units in the plant is written as shown in Table B.3.

B.2 Material description

Materials are described as shown in Table B.4. A generic material represents a type of material in the process, which may exist as one or more specific materials.

<table>
<thead>
<tr>
<th>GENERIC MATERIAL soft.drinks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIFIC MATERIAL lemonade</td>
</tr>
<tr>
<td>SPECIFIC MATERIAL cherryade</td>
</tr>
</tbody>
</table>

Table B.4: Definition language for materials in SUPERBATCH.

B.3 Phase description

A phase is a simple process action which may be performed in the plant, and is independent of any overall process or recipe. The description, shown in Table B.5, starts with a list of the resources and common resources used by the phase. If the phase involves one or more transfers then the route(s) must be described as shown.

In order to be used on-line, the specific unit-phases within each resource phase must be specified. These simply select which unit in each resource will be used, and also specify a
starting string which will be passed to ControlLink when the unit-phase is to be executed in the plant.

```
RESOURCE-PHASE heat_reactor
    USE RESOURCES (reactors)
    USE COMMON RESOURCES (steam)

UNIT-PHASE heat_reactor.1
    RESOURCE reactors UNIT reactor.1
    START SEQUENCE STRING 'heat_reactor.1'

UNIT-PHASE heat_reactor.2
    RESOURCE reactors UNIT reactor.2
    START SEQUENCE STRING 'heat_reactor.2'

ENE RP

RESOURCE-PHASE discharge_reactor
    TRANSFER FROM RESOURCE reactors THROUGH discharge_valves
    TO storage_tank

ENE RP
```

Table B.5: Definition language for phases in SUPERBATCH.

B.4 Recipe description

The recipe definition starts with a header describing details of the recipe, the sizes allowed, and listing the resources and materials which will be used, as shown in Table B.6. The recipe is then defined as a list of steps.

Each step is an instantiation of a resource-phase. It is then given a nominal duration and each transfer is given a size in proportion with the nominal recipe size.

Each resource used by the step is named and a list of preconditions and postconditions may be added. Finally the precedence order between the steps is defined. The step sequence may be linear or branched, and may also contain conditional statements.

B.5 Schedule description

The schedule description begins with a list of each unit and common resource, stating its availability and any initial amounts of material, as shown in Table B.7.

Then the batches may be added as shown in Table B.8. When a batch is added a unique name must be given, together with the recipe that is to be executed. Each generic material mentioned in the recipe description must be specified with a specific material, and likewise each resource must have a unit specified.
Appendix B. SUPERBATCH Models and Input Language

RECIPE make_product
VERSION 2.0
AUTHOR I.K. Brunel
DATE 01/04/2000
BATCH SIZE NOMINAL 100.0
MIN 4.0
MAX 20.0

MATERIAL product
MATERIAL feed

RESOURCE reactors
RESOURCE feed_tanks

COMMON RESOURCE steam

BEGIN

STEP step-1
EXECUTE PHASE charge_reactor
DURATION FIXED AMOUNT 60.0
TRANSFER BATCH SIZE FACTOR 10.0
FROM RESOURCE feed_tank TO reactors
RESOURCE reactors
IN-STATE feed
RESOURCE feed_tank

STEP step-1
EXECUTE PHASE heat_reactor
DURATION FIXED AMOUNT 120.0
RESOURCE reactors
COMMON RESOURCE steam USAGE 2.0
PRECEDING STEP step-1 MAX-WAIT 0.0

END

END RECIPE

Table B.6: Definition language for recipes in SUPERBATCH.

UNIT feed_storage AVAILABLE AT 12:00 01/04/2001
INVENTORY = 100.0 MATERIAL feed

COMMON RESOURCE steam
INITIAL CAPACITY 4.0

Table B.7: Definition language for initial inventories in SUPERBATCH.
When a list of batches is passed to a SUPERBATCH monitor, or a planner running a snapshot, batches may be deleted as well as added, in the way shown.

<table>
<thead>
<tr>
<th>DELETE BATCH ic.batch.001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Recipe make product</td>
</tr>
<tr>
<td>SET MATERIAL feed = feed_A</td>
</tr>
<tr>
<td>RESOURCE reactors UNITS (reactor.1)</td>
</tr>
<tr>
<td>Add BATCH ic.batch.203</td>
</tr>
<tr>
<td>Master Recipe make product</td>
</tr>
<tr>
<td>SET MATERIAL feed = feed_A</td>
</tr>
<tr>
<td>RESOURCE reactors UNITS (reactor.1)</td>
</tr>
<tr>
<td>Add BATCH ic.batch.204</td>
</tr>
<tr>
<td>Master Recipe make product</td>
</tr>
<tr>
<td>SET MATERIAL feed = feed_A</td>
</tr>
<tr>
<td>RESOURCE reactors UNITS (reactor.2)</td>
</tr>
</tbody>
</table>

Table B.8: Definition language for scheduling batches in SUPERBATCH.
Appendix C

Features of Batch Plants Modelled

This appendix describes several plants which have been successfully modelled in the gBOSS++ integrated system. Each plant flowsheet, together with the process STN, is shown.

These features are summarised in the table below.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Series Units (Single-path structure)</td>
<td>Emma</td>
</tr>
<tr>
<td>Parallel Units (Multiple-path structure)</td>
<td>Ruth</td>
</tr>
<tr>
<td>Network Structure</td>
<td>Paula</td>
</tr>
<tr>
<td>Complex shared routing</td>
<td>Nancy</td>
</tr>
<tr>
<td>Recycles</td>
<td>Felicity</td>
</tr>
<tr>
<td>Limited Connectivity</td>
<td>Mel</td>
</tr>
<tr>
<td>Shared Resources</td>
<td>Kathryn</td>
</tr>
<tr>
<td>Semi-continuous processes</td>
<td>Isabel</td>
</tr>
<tr>
<td>Single Products</td>
<td>Mel</td>
</tr>
<tr>
<td>Multiple Products</td>
<td>Julia</td>
</tr>
<tr>
<td>Multiple Recipes for a product</td>
<td>Kathryn</td>
</tr>
<tr>
<td>Cleaning/Product changeovers</td>
<td>Charlotte</td>
</tr>
<tr>
<td>Non-linear processing times</td>
<td>Paula</td>
</tr>
<tr>
<td>Sequential transfers</td>
<td>Emma</td>
</tr>
<tr>
<td>Simultaneous transfers</td>
<td>Sarah</td>
</tr>
<tr>
<td>Multiple transfers in a step</td>
<td>Mel</td>
</tr>
<tr>
<td>No Intermediate Storage (Stable states)</td>
<td>Paula</td>
</tr>
<tr>
<td>Limited Intermediate Storage</td>
<td>Sarah</td>
</tr>
<tr>
<td>Unlimited Intermediate Storage</td>
<td>Julia</td>
</tr>
<tr>
<td>Multiple Storage Tanks</td>
<td>Patricia</td>
</tr>
</tbody>
</table>

Table C.1: Features of Batch Processing which can be modelled.
Charlotte

This plant consists of three processing units in series, as shown below.

![Figure C.1: Flowsheet for Charlotte's Plant](image)

There are two products, each of which travel through the three processing units. In addition there is a cleaning recipe which must be executed every time the product changes.

![Figure C.2: STN for Charlotte’s Plant](image)
Appendix C. Features of Batch Plants Modelled

Emma

This plant consists of three processing units in series, as shown below.

![Flowsheet for Emma's Plant](image)

Figure C.3: Flowsheet for Emma’s Plant

There are two products, each of which travel through the three processing units. Basically this is the same as Charlotte but without the cleaning between products.

![STN for Emma's Plant](image)

Figure C.4: STN for Emma’s Plant
Appendix C.  Features of Batch Plants Modelled

Felicity

This plant includes recycled material.

If the second processing step is carried out slowly then the solvent can be reused, otherwise it is discarded.
Ingrid

![Diagram of Ingrid's Plant](image)

**Figure C.7: Flowsheet for Ingrid's Plant**

The canning and drumming lines are taken to be separate units with their own internal storage.

![Diagram of STN for Ingrid's Plant](image)

**Figure C.8: STN for Ingrid's Plant**

The reaction task has a number of states entering and leaving during the task.
Isabel
This plant is of a network structure, as shown below.

Heat_A is carried out in R1, React_1 in R2, React_2 in R3, React_3 in R1 and React_4 in R4.

The processing times are non-linear functions of the batch size.
Appendix C. Features of Batch Plants Modelled

Figure C.11: Flowsheet for APCI Plant (non-confidential version)
Kathryn
This is a model of the batch pilot plant.

Figure C.12: Flowsheet for the Batch Pilot Plant

Figure C.13: STN for the Batch Pilot Plant
Appendix C. Features of Batch Plants Modelled

Mel
Melinar paste plant.

Figure C.14: Flowsheet for Mel’s Plant

Figure C.15: STN for Mel’s Plant
Appendix C. Features of Batch Plants Modelled

Nancy

![Flowsheet for Nancy's Plant](image1)

Figure C.16: Flowsheet for Nancy's Plant

![STN for Nancy’s Plant](image2)

Figure C.17: STN for Nancy’s Plant
Appendix C. Features of Batch Plants Modelled

Patricia

Figure C.18: Flowsheet for Patricia's Plant
Figure C.19: STN for Patricia’s Plant
Appendix C. Features of Batch Plants Modelled

Paula

The aim of this plant is to show how semi-continuous processes may be modelled. A reactor produces batches of material, which are stored in a buffer tank. This is then drained continuously by a steam stripper.

Figure C.20: Flowsheet for Paula’s Plant

Figure C.21: STN for Paula’s Plant
Appendix C. Features of Batch Plants Modelled

Ruth

This plant consists of a network of three mixers and three reactors, as shown below.

![Flowsheet for Ruth's Plant](image)

Figure C.22: Flowsheet for Ruth's Plant

Three products are made, each with one stage in a mixer and one in a reactor.

![STN for Ruth's Plant](image)

Figure C.23: STN for Ruth's Plant
Sarah

This is a simple example of a plant with limited intermediate storage. The plant consists of four raw material tanks, two product tanks, a reactor, a blender and an intermediate storage tank.

Two feeds, A and B, are reacted to form a stable intermediate C. This is stored in an intermediate storage tank. From here it is taken and blended with one of two additives depending upon which product is to be made.
Appendix C. Features of Batch Plants Modelled

Tracy

The plant consists of two main reactors with a number of raw material, intermediate and final product storage tanks.

![Flowsheet for Tracy's Plant](image)

Figure C.26: Flowsheet for Tracy's Plant

This process consists of a number of reaction stages, with stable and unstable intermediates, as well as the recycle of unreacted materials.

![STN for Tracy's Plant](image)

Figure C.27: STN for Tracy's Plant
Appendix C. Features of Batch Plants Modelled

Carlene
This plant consists of a network of three process units and ten storage units.

Figure C.28: Flowsheet for Carlene’s Plant

Five products are made, each with a different path though some or all of the processing units.

Figure C.29: STN for Carlene’s Plant