MODELLING AND ANALYSIS OF AGENT BASED DISTRIBUTED SCHEDULING SYSTEMS

by

Mahmut Kurşun

BS. in M.E., Boğaziçi University, 1997

MS. in I.E., Boğaziçi University, 2001

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Industrial Engineering

Boğaziçi University

2008

ACKNOWLEDGEMENTS

I would like to express my gratitude to Ali Tamer Ünal, for his continuous support and his indispensable advices throughout my study. Within the last 10 years of my graduate studies it was a real chance and a pleasure for me to work with him. He never stopped encouraging me even at hardest times, and without his advices not limited to academic work, it could not be possible to complete this work.

I also like to thank to Emre Otay who consistently monitored my study, and was always available with his challenging and insightful comments, follow-ups, and critical reading of this thesis and suggestions throughout this work.

I also would like to thank to Tülin Aktin, Necati Aras, and Taner Bilgiç, for their trust on me during this study and for their valuable comments.

I also would like to thank to Kamer Sözer for her valuable comments and timeless discussions throughout this study.

I would like to thank also to Nurhan Kalfayan, for being so kind to host the two noisy quad core servers at his home, and for consistently monitoring their uptime status.

I would like to thank to Turgay Ciner, for supporting my graduate studies and for his patience during the last 10 years.

No words I can find to express my thanks to my family, especially to my mother for her unbelievable support, and indefinite patience during the years of my thesis study. Without her continuous support and patience I could not finish this study.

To my mother, and the memory of my grandmother and my grandfather …

ABSTRACT

MODELLING AND ANALYSIS OF AGENT BASED DISTRIBUTED SCHEDULING SYSTEMS

In this study we have introduced a mathematical foundation and a framework for the modeling and analysis of agent based distributed scheduling systems. We defined the basic structure of an object oriented agent based system, and the issues in distributed systems. In order to explore the agent based structures deeply, a state based modeling approach is used.

We combine a set of decision variables or control variables under the name of control policy. The control policy is comprised of the decision process (DP), response policy (RP), and agent stability during the decision process. To test the impact of various control policies to the performance of the system we set up an experiment with different problem settings based on due date tightness, the frequency of job arrivals, and processing time distribution.

In this study we have been able to show that the solution procedure by itself does not guarantee a good overall performance in the system when the control policy is not set up correctly. Also we have been able to show the performance of these control policies change from one problem setting environment to another. So it is not possible to say that one magic solution procedure will be able to solve all the problem sets. It has been showed that introducing the decision time to the decision process leads to significantly poorer results compared to instantaneous policies, and even a responsive random dispatch policy may perform well better compared to a takes time control policy equipped with an optimum dispatch algorithm.

ÖZET

AJAN TABANLI DAĞITIK ÇİZELGELEME SİSTEMLERİNİN ANALİZİ VE MODELLENMESİ

Biz bu çalışmada ajan tabanlı dağıtık çizelgeleme sistemlerinin modellenmesi ve analizi için bir yazılım altyapısı ve matematiksel formülasyon sağladık. Nesne yönelimli ajan tabanlı sistemlerin altyapısını tanımladık, ve dağıtık sistemlerde gözlenen problemlere değindik. Ajan bazlı sistemleri derinlemesine incelemek için durum tabanlı modelleme tekniğini kullandık.

Karar veya kontrol değişkenlerinden oluşturduğumuz seti kontrol politikası olarak adlandırdık. Bu kontrol politikasını, karar süreci, tepki politikası ve karar süreci sırasındaki ajan stabilitesinin kombinasyonundan oluşturduk. Değişik kontrol politikalarının sisteme etkilerini test etmek amacıyla bir deney ortamı hazırladık. Bu deney ortamında termin tarihlerinin sıkılığı, işlerin gelme frekansı, işlem zamanının dağılımına gore problemleri klasifiye ve test ettik.

Bu çalışma sonucunda kontrol politikaları doğru kurulmadıysa çözüm metodolojisinin tek başına iyi sonuçları garanti edemeyeceğini gösterdik. Aynı zamanda bu kontrol politikalarının farklı problem tipleri için farklı performanslarla çalıştığını tespit ettik. Karar prosesinin zaman alan bir politika olması durumunda karar zamanı içeren politikaların anlık cevap veren politikalara gore daha kötü sonuçlar verdiğini, hatta tamamen rassal çözüm üreten anlık bir çözüm politikasının bile karar vermesi zaman alan ama optimum çözüm üreten bir politikadan daha iyi sonuçlar verebildiğini gösterdik.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

ABSTRACT iv

ÖZET v

LIST OF FIGURES x

LIST OF TABLES xviii

LIST OF SYMBOLS / ABBREVIATIONS xix

1. INTRODUCTION 1

2 . LITERATURE REVIEW 4

2.1. Static Deterministic Problems 5

2.1.1. Centralized Solutions 5

2.1.2. Distributed Solutions 5

2.2. Static Stochastic Problems 8

2.2.1. Centralized Solutions 8

2.2.1.1. Continuous Review 8

2.2.1.2. Simulation Based 9

2.2.2. Distributed Solutions 10

2.3. Dynamic Stochastic Problems 11

2.3.1. Distributed Solutions 11

2.4. Review Papers 12

2.5. Communication 13

2.5.1. Blackboard Model 15

2.5.2. Message Passing 15

3. BASIC STRUCTURE OF AN OBJECT ORIENTED AGENT BASED SYSTEM 17

3.1. Structure of the System 17

3.2. Behaviour of the System 21

3.2.1. Message to Event Processor (MEP) 23

3.2.2. Dispatch Function 25
3.2.3. Activate Statechart function: ActivateSC 25
3.2.3.1 . Algorithm: ActivateSC (o, SCoi , e, t) 26
3.2.4. Action 26
3.3. Messaging Mechanism 27
4. ISSUES IN DISTRIBUTED SYSTEMS 30
4.1. Simultaneous Transformation Request Problem 34
4.2. Observability Problem 34
4.3. Conflict of Interest of Agents 35
4.4. Robust Representation 36
5. AGENT BASED MODELING OF DYNAMIC SCHEDULING SYSTEM 37
5.1. Basic Definitions 37
5.2. Working of the Agent System in Terms of the Mathematical Model 40
6. EXPERIMENTAL SETUP 43
6.1. Problem Setting Parameters 44
6.2. Control Policy Parameters 47
6.3. Analyzed Problems 49
6.4. Random Problem Generation 50
6.4.1. System Setup 51
6.4.2. Job Interarrival 52
6.4.3. Tightness of Due Dates 52
6.4.4. Decision time 53
6.4.5. Machine Breakdowns 54
6.4.6. Response Policy 54

7. RESULTS 56

7.1. Effect of Decision Time under IE based SF policy 58
7.2. Effect of Decision Time under IE based JS policy 66
7.3. Effect of Decision Time under TES based SF Policy 69
7.4. Effect of Decision Time under TEU based SF Policy 76
7.5. Effect of the Rescheduling Interval under IE based SF Policy 83
7.6. Effect of the Rescheduling Interval under TES based SF Policy 84
7.7. Effect of the Rescheduling Interval under TEU based SF Policy 86
7.8. Solution Procedure vs. Control Policies 88
7.8.1. The Random Baseline Solution Procedure : IE RND 94
7.9. Effects of Control Policies at a Glance 94

8. CONCLUSION 96

APPENDIX A. ICRON IMPLEMENTATION 99

A.1. ICRON as the Preferred Framework 99
A.2. State chart Implementation in Icron 101
A.2.1. Aspects of Objects as State charts 106
A.3. MEP Implementation in Icron 110
A.3.1. Activate State chart Node 111
A.3.1.1. Working of the ActivateSC Function 112
A.3.2. Dispatch Message Node 113
A.3.3. Processing of System Messages 114
A.4. DISPATCH Implementation in Icron 116
A.4.1. Send to Agent Node 117
A.4.2. Dispatch Message Algorithm 118
A.4.3. SC_MSG Algorithm 119
A.5. Messaging Mechanism in Icron 119
A.5.1. Handle Message Algorithm 121
A.5.2. Handle Dispatch Message Algorithm 122

APPENDIX B. DISTRIBUTED SIMULATOR IN ICRON 125

B.6. Events and Event Handling in ICRON 128
B.6.1. Processing of Events 130

B.6.1.1. Handle Algorithm of the Breakdown Event 131
B.6.1.2. Handle Algorithm of the New Job Event 132
B.6.1.3. Handle Algorithm of the Repair Completion Event 133
B.6.1.4. Handle Algorithm of the Operation Completion Event 134
B.6.1.5. Handle Algorithm of the Delta Event 135
B.7. Messaging Flow of the Simulation Events in ICRON 139
B.7.1. Messaging Flow for Breakdown simulation Event 139
B.7.2. Messaging Flow for New Job Event 143
B.7.3. Messaging Flow for Repair Completion Event 149
B.7.4. Messaging Flow for Operation Completion Event 154
B.8. Messaging in ICRON 161
B.8.1. BREAKDOWN Message Declaration 162
B.8.2. NEWJOB Message Declaration 162
B.8.3. REPAIRCOMPLETION Message Declaration 165
B.8.4. OPERATIONCOMPLETION Message Declaration of the Simulator Agent 165
B.8.5. REGISTER_DELTA_TIME Message Declaration 166
B.8.6. DELTA_MSG Message Declaration 167
B.8.7. CS Message Declaration 167
B.8.8. OPERATIONSTART Message Declaration 168
B.8.9. OPERATIONSTOP Message Declaration 168
B.8.10. OPERATIONCOMPLETION Message Declaration of the Shop Floor Renewable Resource Agent 169
B.8.11. JOBCOMPLETED Message Declaration 169
B.8.12. CENTRAL_SCHEDULE Message Declaration 170
B.8.13. SIMULATIONCOMPLETED Message Declaration 170
REFERENCES 172
REFERENCES NOT CITED 178

LIST OF FIGURES

Figure 2.1. Overview of the scheduling problems in the literature 4

Figure 2.2. Classic model of theory of communication..... 14

Figure 3.1. Working of an agent 22

Figure 3.2. Working of a Proxy Agent 23

Figure 3.3. ActivateSC algorithm 26

Figure 3.4. The messaging flow among agents 28

Figure 6.1. Single Machine, Two Agents Control Scenario 44

Figure 6.2. Sample arrival of jobs..... 53

Figure 6.3. Decision Time effect on the arriving jobs 54

Figure 7.1. Results under different JS Control Policies and IE based SF control policy..... 59

Figure 7.2. The TES-IE policy results vs. Decision Times 61

Figure 7.3. The TEU-IE policy results vs. Decision Times..... 62

Figure 7.4. TCS control policy at CS agent, and the dispatch intervals, where x represents the mean job interarrival time, and Δ the CS delta time 63

Figure 7.5. TCS vs. TCU JS Policy under IE SF policy under different CSR values 64

Figure 7.6. TCS-IE Decision Time Variation under different CS Rescheduling intervals 65

Figure 7.7. TCU-IE Decision Time Variation under different CS Rescheduling intervals 65

Figure 7.8. SF Control Policy variations under instantaneous and event based job system policy 67

Figure 7.9. IE-TES control policy results vs. Decision Time 68

Figure 7.10. IE-TEU control policy results vs. Decision Time 68

Figure 7.11. JS Control Policy variations under SF=TES with LS Delta=0.5 70

Figure 7.12. JS Control policy variations under SF=TES with CS Delta=0.5 and LS Delta=0.5 71

Figure 7.13. JS Control policy variations under SF=TES with CS Delta=0.1 and LS Delta=0.5 72

Figure 7.14. Decision Time variations under TES-TES control policy 73

Figure 7.15. Decision Time variations under TEU-TES control policy 74

Figure 7.16. Decision Time variations under TCS-TES control policy 74

Figure 7.17. Decision Time variations under TCU-TES control policy 75

Figure 7.18. Decision Time Variations under IC-TES 76

Figure 7.19. JS Control policy variations in Loose Due Date settings at SF=TEU, 0.5 LS Delta 77

Figure 7.20. JS Control policy variations in Tight Due Date settings at SF=TEU, 0.5 LS Delta 78

Figure 7.21. JS Control policy variations under SF=TEU with CS Delta=0.5 and LS Delta=0.5 78

Figure 7.22. JS Control policy variations under SF=TEU with CS Delta=0.1 and LS Delta=0.5 79

Figure 7.23. Decision Time variations under TES-TEU control policy 80

Figure 7.24. Decision Time variations under TEU-TEU control policy 81

Figure 7.25. Decision Time variations under TCS-TEU control policy 81

Figure 7.26. Decision Time variations under TCU-TEU control policy 82

Figure 7.27. Decision Time variations under IC-TEU control policy 82

Figure 7.28. Effect of the CS Rescheduling Interval on IC-IE control policy 83

Figure 7.29. Effect of the CS Rescheduling Interval on TCS-IE control policy 84

Figure 7.30. Effect of the CS Rescheduling Interval on IC-TES Policy 85

Figure 7.31. Effect of the CS Rescheduling Interval on TCS-TES Policy 85

Figure 7.32. Effect of the CS Rescheduling Interval on TCU-TES Policy 86

Figure 7.33. Effect of the CS Rescheduling Interval for IC-TEU policy 87

Figure 7.34. Effect of the CS Rescheduling Interval for TCS-TEU policy 87

Figure 7.35. Effect of the CS Rescheduling Interval for TCU-TEU policy 88

Figure 7.36. IE RND compared to other SF control policy variations under JS=IE 90

Figure 7.37. IE RND compared to other SF control policy variations under JS=IC 90

Figure 7.38. IE RND compared to other SF control policy variations under JS=TES 91

Figure 7.39. IE RND compared to other SF control policy variations under JS=TEU ... 92

Figure 7.40. IE RND compared to other SF control policy variations under JS=TCS... 93

Figure 7.41. IE RND compared to other SF control policy variations under JS=TCU .. 93

Figure 7.42. Control policy variations at a glance 95

Figure A.1. A view of the development tab in Icron 103

Figure A.2. The state chart of the machine object in the shop floor 104

Figure A.3. The state chart of an operation object in the shop floor 105

Figure A.4. SF_OPERATION_DB, and SF_RR_DB state charts of an operation and a renewable resource object in the shop floor 106

Figure A.5. Sfs_DB State chart of the shop floor system object. 108

Figure A.6. The state chart of the SIM renewable resource in the SFSim domain 109

Figure A.7. State chart of the SCH Renewable Resource object..... 110

Figure A.8. Activate State Chart Node 111

Figure A.9. Dispatch message node..... 113

Figure A.10. A sample MEP algorithm for handling of a BREAKDOWN message in the shop floor 115

Figure A.11. Send to agent node..... 117

Figure A.12. Sample DISPATCH algorithm for handling the “BREAKDOWN” and “REPAIRCOMPLETION” messages in the shop floor..... 119

Figure A.13. A view of the channels in the development tab in Icron 120

Figure A.14. The generic HandleMessage algorithm for listening the messaging channels..... 122

Figure A.15. DispatchMessage algorithm, the relevant dispatch algorithm finder 123

Figure B.1. Overview of the System Components of IDM 126

Figure B.2. The “Handle” algorithm of the main event class 130

Figure B.3. Handle Algorithm of the Event BREAKDOWN 131

Figure B.4. Handle algorithm of the event NEWJOB 133

Figure B.5. Handle algorithm of the event REPAIRCOMPLETION 134

Figure B.6. Handle Algorithm of the Event OPERATIONCOMPLETION 135

Figure B.7. Handle Algorithm of the Event Delta 136

Figure B.8. DeltaMessage algorithm 137

Figure B.9. A sample MEP algorithm for handling of a CS message 138

Figure B.10. Breakdown event message flow diagram 139

Figure B.11. MEP-BREAKDOWN algorithm for Simulator Agent 140

Figure B.12. DISPATCH_Machine_MSG algorithm 141

Figure B.13. MEP_BREAKDOWN algorithm in the SF_RENEWABLE RESOURCE_AGENT 142

Figure B.14. SF Renewable Resource DISPATCH Breakdown algorithm 142

Figure B.15. New Job event message flow diagram 143

Figure B.16. MEP-NEWJOB algorithm for Simulator Agent 144

Figure B.17. DISPATCH_NEWJOB_MSG algorithm 144

Figure B.18. MEP-NEWJOB algorithm in the SCH_JOB_AGENT 145

Figure B.19. MEP-NEWJOB algorithm of the Schedule_Agent 148

Figure B.20. DISPATCH algorithm of the Schedule Agent for NEWJOB and DISPATCH_TO_LOCAL_SCHEDULERS messages..... 149

Figure B.21. MEP-NEWJOB algorithm of the SCH_RENEWABLE_RESOURCE_AGENT 149

Figure B.22. Repair Completion event message flow diagram 150

Figure B.23. MEP-REPAIRCOMPLETION algorithm of the simulator agent 151

Figure B.24. MEP-REPAIRCOMPLETION algorithm of the SF_RENEWABLE_RESOURCE AGENT 152

Figure B.25. DISPATCH-REPAIRCOMPLETION of the SCH_RENEWABLE_RESOURCE_AGENT 153

Figure B.26. MEP-OPERATIONCOMPLETION of the SCH_RENEWABLE_RESOURCE_AGENT 154

Figure B.27. Operation Completion event message flow diagram..... 156

Figure B.28. MEP-OPERATIONCOMPLETION algorithm of the Simulator Agent... 156

Figure B.29. MEP-OPERATIONCOMPLETION algorithm of the SF_RENEWABLE_RESOURCE AGENT 157

Figure B.30. DISPATCH-OPERATIONCOMPLETION algorithm of the SF_RENEWABLE RESOURCE_AGENT 158

Figure B.31. MEP-OPERATIONCOMPLETION of the SCH_JOB_AGENT..... 158

Figure B.32. DISPATCH-OPERATIONCOMPLETION algorithm of the SCH_JOB_AGENT 159

Figure B.33. MEP-JOBCOMPLETED of the Simulator Agent..... 160

Figure B.34. DISPATCH-SIMULATIONRUNCOMPLETED of the Simulator Agent..... 160

Figure B.35. MEP-SIMULATIONCOMPLETED algorithm of the SCH_JOB_AGENT 161

Figure B.36. Representation of the relationships of a new job and its building blocks . 163

LIST OF TABLES

Table 6.1. All four possible combinations of the problem setting parameters 46

Table 6.2. The control policy parameters at one glance 48

Table 7.1. LS and CS agents control policies 56

Table 7.2. The Difference Values in terms of Job Interarrival Times for selected control policies under different problem setting parameters, values for Figure 7.1 60

LIST OF SYMBOLS / ABBREVIATIONS

A	A set of agents
$B_{im}(\omega)$	random variable B_{im} corresponding to ω
b_{im}	the i^{th} breakdown time for machine m
C_q	completion time of operation q
c	control policy
$D_{im}(\omega)$	random variable D_{im} corresponding to ω
d_{im}	the repair duration of i^{th} breakdown for machine m
E	Finite set of events
E^a_l (c)	set of events corresponding to message type l interpreted by agent a based on control scenario c
F_q	start time of operation q
f_e (c)	a transformation function in response to event e
$g_{il}^a(\omega, c)$	time of the i^{th} message of type l received by agent a
L^a	set of messages that can be received by agent a
l	message type
$I_j(\omega)$	random variable I_j corresponding to ω
J	A set of jobs
J^a	A subset of the jobs, which are to be controlled by agent a ε A.
M	A set of machines
M^a	A subset of machines, which are to be controlled by agent a ε A.
M_q	assigned machine of operation q
N^o_i:	the named states of state chart i of object o.
O	Finite set of objects
Q_j	A set of operations of job j ε J.
Qj^a	All operations of the jobs J^a , which are to be controlled by agent a ε A.
o	object o
$P_{qm}(\omega)$	random variable P_{qm} corresponding to ω
p_{qm}	the processing time of operation q on machine m
R^S	State Event relation

R^E	Event Action relation
r_j	ready time of job j
S	Finite set of states.
$S_{jt}^a(\omega,c)$	state of job j at time t observed by agent a according to realization ω and c
$S_{mt}^a(\omega,c)$	state of machine m ε M^a at time t observed by agent a according to realization ω and c
$S_t^o(\omega, c)$	the state of the object o at time t according to the global realization ω, and to the control policy c.
$S_{qt}^a(\omega, c)$	state of operation q at time t observed by agent a according to realization ω and control policy
SC^o_i	the state chart i of object o ε O
t	time index
u	problem setting parameter
V_m	status of the machine m
X_q	status of being processed or not of operation q
Y_q	percent completion of operation q
Z	set of actions.
Δ	duration time
ϕ	another duration time
ω	An elementary realization
Θ	probability space
Ω	An action
Φ	state space
ACL	Agent Communication Language
AGV	Automated Guided Vehicle
CNC	Computer Numerical Control
CS Agent	Central Scheduling Agent
GSAMS	Graphical Scheduling Algorithm Modeling System
FIPA	Foundation for Intelligent Physical Agents
JS	Job System

LH	Loose Due Date and High Utilization
LL	Loose Due Date and Low Utilization
LS Agent	Local Scheduler Agent
IC	Instantaneous Cyclic
IE	Instantaneous event based
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
P2P	Point to point
RAD	Rapid Application Development
RSAD	Rapid Scheduling Application Development
SF	Shop Floor
TCS	Takes Time Cyclic and Stable
TCU	Takes Time Cyclic and Unstable
TES	Takes Time Event Based and Stable
TEU	Takes Time Event Based and Unstable
TH	Tight Due Date and High Utilization
TL	Tight Due Date and Low Utilization
XML	Extensible Markup Language

1. INTRODUCTION

Production scheduling is primarily focused on finding optimal, or near-optimal, schedules with respect to various criteria. These approaches use the implicit assumption of a static deterministic environment where complete knowledge of the problem is available without consideration of any kind of failures. Unexpected events in the shop floor, like machine failures, urgent job arrivals, job cancellations (Vieira et al. 2003), not only interrupt system operation but also upset the predictive schedule that was previously established. Consequently the resulting schedule may neither be feasible nor nearly optimal anymore.

Lawrence and Sewel (1997) empirically show that performance of "optimized" static schedules may deteriorate rapidly with processing time uncertainty and that simple dynamic dispatching heuristics may provide a far superior performance. Kutanoglu and Sabuncuoglu (2001) report similar observations when considering uncertainties in job processing times and machine availability.

Another observation in industrial scheduling settings is that, due to the complexity of scheduling problems, centralized systems fail to timely respond to unexpected events. To handle the dynamic nature of the problem in a responsive manner, one of the proposed solutions is to decompose the problem and distribute to various scheduling servers. This approach is called distributed scheduling (Jeong and Leon 2005, Kutanoglu and Wu 2004, Attanasio et al. 2006, Chun and Wong 2003, Tharumarajah 2001, Cavalieri et al. 2000, Pendharkar 1999, Liu and Sycara 1997, Baker 1998, Wang et al. 2003).

In distributed scheduling, a number of schedulers, i.e. scheduling agents, observe a specific part of the decomposed system, and generate solutions. Depending on the distribution architecture, schedulers may communicate between each other. In many scheduling environments, even the decomposed scheduling problem is NP-hard, and it takes significant time for the scheduler to make a decision. In this time frame , which we call the decision-time, the scheduler is in the process of running a scheduling algorithm, hence information it has is incomplete. An agent either ignores all the events it observes

during the decision-time, which we call an unstable agent, or it responds to events using a previously generated schedule, which most probably was based on an earlier state of the system, which we call a stable agent.

Throughout this study we will be dealing with online scheduling problems, i.e. there will be new job arrivals or machine breakdowns in the system. We will always be dealing with dynamic problems with no prior information about the future events. To analyze the complexity of the proposed problem we will use a single machine problem. To clarify the complexity of the problem, assume that we will be dealing with a static version of the mentioned problem with all new job arrivals known a-priori with finite set of jobs. The objective throughout this study will be to minimize the maximum lateness, i.e. Lmax, or the average lateness. So our problem will be a 1| rj | Lmax type of problem, and this problem without preemption is a strongly NP-hard problem. Since the simplest version of the problem is strongly NP-hard, the online version without any prior information about the new job arrivals will be at least as hard as the static version.

Another important point that is not very well studied in the literature is the representation problem of the system. In real life manufacturing systems there are a lot of unexpected events occurring almost every time, as Vieira et al. (2003), summarized them. On the other hand in real life manufacturing systems the data gathering process itself is a problematic process. At the instant the shop floor information is gathered, and scheduling of the system according to these data is tried to be performed, a machine may breakdown or a new very urgent job with high penalty cost may arrive to the system. When the recently generated schedule is ready to be implemented to the system, the whole problem may change. So even when the previous problem is solved and an optimal solution is reached, it might not be valid for the new current situation of the shop floor. Therefore one may experience an observability problem which we will discuss in Section 4.2.

In this study, we analyze a dynamic distributed single machine scheduling system. We especially investigate the impact of the time it takes schedulers to make a decision, i.e. the scheduling process, on the system performance. It is obvious that, one would always choose a system where the time it takes to make a decision is as small as possible. However, the trade-off we explore is between the length of the decision-time and the

quality of the decisions. Generally, we would expect to make better decisions if we spend more time. But, this would, on the other hand, make the process less responsive. Until now, every study in the literature assumed that just instantaneously any strongly NP-hard scheduling problem will be solved, and a solution to this problem will be reached instantaneously. But unfortunately that can not be the case, because of the complex nature, and time complexity of the problem itself.

In this study we propose a state based modeling approach to represent the working of a scheduling agent so that effect of choices made in distributing the problem in terms of the overall performance of the scheduling methodology can formally be defined and analyzed. So we will try to analyze all the common issues in distributed systems like simultaneous transformation request problem and observability problem.

In the next chapter, an extensive literature review is presented. After we will analyze the basic structure of an object oriented agent based system. Then we will discuss the issues in distributed systems, and then continue with the agent based modeling of dynamic scheduling systems. We will continue with the experimental setup. The Results and Conclusion sections will complete the study. The Icron implementation and analysis of the distributed simulator implemented in Icron will be presented in Appendix A and B.

2. LITERATURE REVIEW

We first categorize the scheduling problems as *static* and *dynamic* problems. Problems with finite set of jobs will be indicated as Static problems, whereas problems with infinite set of jobs will be defined as Dynamic problems. After this categorization, we further divide both types of problems in to two, as *deterministic*, and *stochastic*. In the deterministic case, all information is given, whereas in the stochastic case, some information is uncertain. In the stochastic case new job arrival rate, machine failure rate, processing time may be a stochastic distribution. Not limited to these, the most common factors identified in the literature as being unexpected events in the shop floor are as the followings; machine failures, urgent job arrivals, job cancellation, due date changes, delay in the arrival or shortage of materials, change in job priority, over- or underestimation of processing times.

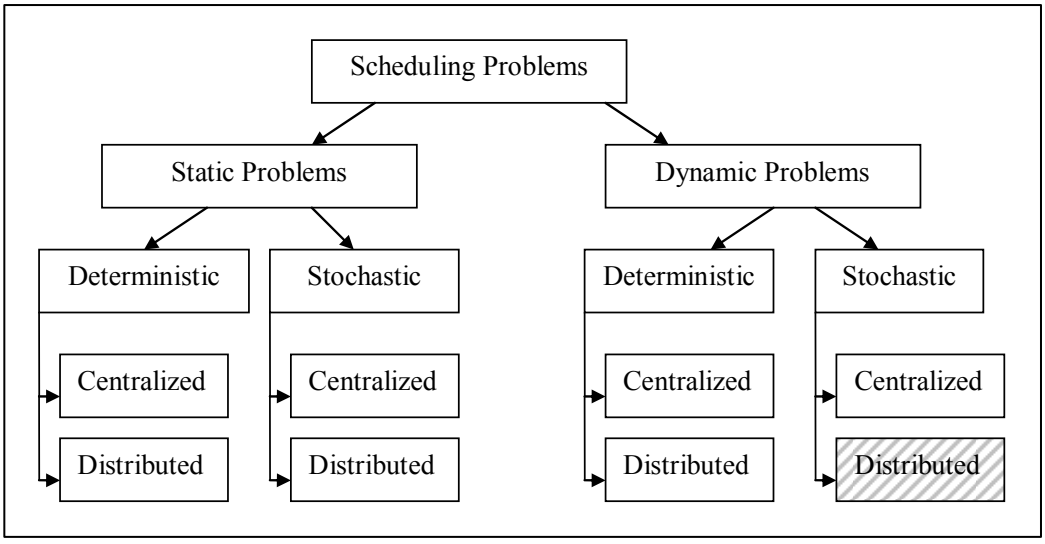

Figure 2.1. Overview of the scheduling problems in the literature

On top of these definitions we finally add the last layer as the solution methodology layer, and categorize it as *centralized*, and *distributed* solution methodologies. Our main interest area is the dynamic stochastic problems with distributed solution methodologies. When we are talking about distributed solution methodologies, then agent infrastructures are very common. With introduction of agent infrastructures there are also a lot of issues

regarding communication protocols, negotiation frameworks, reliability, scalability and robustness. For this reason after we review the types of scheduling problems, we will also focus on agent communication and negotiation literature.

2.1. Static Deterministic Problems

2.1.1. Centralized Solutions

Jeong and Leon (2005), studied a special case of a single machine problem, where the decision authorities and information are distributed in multiple sub-production systems, and these sub-production systems share the single machine, and must cooperate with one another to achieve a global goal of minimizing a linear function of the completion times of the jobs; e.g., total weighted completion times. It is assumed that neither the sub-production systems nor the shared machine have complete information about the entire system. The associated scheduling problems are formulated as zero-one integer programs. Their solution approach is based on Lagrangian relaxation techniques which are modified to require less global information. Specifically, there is no need for a global upper bound, or a single master problem that has a complete view of all the coupling constraints. Experimentally Jeong and Leon (2005), showed that the proposed methodology exhibits a promising performance compared to the Lagrangian relaxation with a subgradient method with the added benefit that can be applied to situations with more restrictive information sharing.

2.1.2. Distributed Solutions

Wang (2003), proposed a distributed adaptive scheduling (DAS) approach to achieve collective meeting scheduling for groupwares. Wang (2003), showed that DAS can quickly find suitable meeting solutions for a large number of users, by utilizing a set of flexible co-ordination agents. The co-ordination agents can move from one machine to another to share information and to co-ordinate a mutually acceptable meeting solution. They can also make instant decisions on the feasibility of proposed time-slots according to currently collected information. As a consequence quick and effective scheduling can be achieved with good privacy maintenance for users distributed at different places. Simulated

experiments have shown that DAS performs well in communication cost and privacy protection when compared with traditional scheduling methods, especially when scheduling with a large number of people and when scheduling with disparate user schedules.

Karageorgos et. al. (2003), present an agent-based approach for supporting logistics and production planning, taking into account not only production schedules but also availability and cost of logistic service providers. In open virtual enterprise collaboration networks, manufacturing and logistics planning and scheduling are challenging due to the difficulties in integrating information from different partners and in exploring a large and dynamically changing number of planning and scheduling alternatives. Agent-based technology is considered suitable to support planning and scheduling in such enterprises because agents can dynamically adapt their behaviour to changing requirements and they can reduce the number of planning and scheduling alternatives via negotiation.

Kutanoglu and Wu (2006), studied the incentive compatible, collaborative production scheduling with distributed agents. They handle the case of asymmetric information for the independent agents, which may not share private information about their state. So they address the issue of incentive compatibility to align their behavior in favor of overall system efficiency. Specifically, they design a 'schedule selection game' where all participating agents state their preferences via a valuation scheme, and the mechanism selects a final schedule based on the collective input. Kutanoglu and Wu (2006), contributed to the existing literature by studying a more general case of a multi-stage, multi-machine environment with a set of pre-generated candidate schedules. Instead of resource agents competing to win job requests, they examined the case where job agents compete for scarce resources (machine–time slots).

Chun et. al. (2003), studied agent based meeting scheduling through preference estimation. Meeting scheduling is a routine task that needs to be performed quite regularly and frequently within any organization, which can be quite tedious and time-consuming, potentially requiring a several rounds of negotiations among many people on the meeting date, time and place before a meeting can finally be confirmed. They create an agent-based environment within which meeting scheduling can be performed and optimized. For

meeting scheduling, they define optimality as the solution that has the highest average preference level among all the possible choices. Their model tries to mimic real life in that the individual's preferences are not made public. Without complete information, traditional optimal algorithms will not work. In this work, they present the new ''preference estimation'' technique that allows them to find optimal solutions to negotiation problems without needing to know the exact preference models of all the meeting participants beforehand. Instead, their preferences are ''estimated'' and built on the fly based on observations of their responses during negotiation. Another contribution is the use of ''preference rules'' that allow preferences to change dynamically as scheduling decisions are made. This mimics changing preferences as schedule gets filled.

Cesta and D'aloisi (1999), studied mixed-initiative interaction between users and agents and proposed an agent system for distributed meeting scheduling, and explained the solutions developed to control interaction. One of the main contributions that are not studied before is the study why a user should trust the user, and when will be the responsibility of the decision is up to the user.

Attanasio et al. (2006), presented a preliminary study which aims at developing auction mechanisms for decentralized scheduling which exhibit minimal communication overhead and an efficient usage of resources. Their computational results have shown that an auction mechanism based on a progressive Lagrangean heuristic is able to provide results comparable to those provided by centralized heuristics.

Kim and Paulson (2003), present an agent-based compensatory negotiation methodology to facilitate the distributed coordination of project schedule changes where a project can be rescheduled dynamically through negotiations by all of the concerned subcontractors. The methodology consists of a compensatory negotiation strategy based on utility of timing, multi-linked negotiation protocols, and message-handling mechanisms. In practice, this methodology may improve project network schedules, by lowering the sum of the cost of the subcontractors associated with their resource constraints. In theory, the methodology provides a distributed coordination methodology that can improve interaction and collaboration among agents and people.

Chun and Wong (2003), presented a distributed negotiation framework and their negotiation algorithm. The framework consists of a user preference model and an approach that makes use of the model to evaluate proposals and suggest counter proposals, using a preference level measure. The negotiation process is guided by a proposal evaluation function that evaluates the global preference level for a particular proposal. The negotiation algorithm finds the optimal solution that maximizes average preference levels, and is a general negotiation algorithm, and can also be used for resource allocation problems. They used a special case of this framework and the algorithm on the classical meeting scheduling problem to perform agent-based meeting scheduling.

2.2. Static Stochastic Problems

2.2.1. Centralized Solutions

2.2.1.1. Continuous Review: Sabuncuoglu and Bayız (2000), reviewed the reactive scheduling problems and compared them according to several criteria in both deterministic and stochastic test environments. They test several scheduling policies under machine breakdowns in a classical job shop system. In addition, they measured the effect of system size and type of work allocation (uniform and bottleneck) on the system performance. They evaluated the performance of the system for the mean tardiness and makespan criteria. They found out that partial scheduling with optimization based scheduling algorithms can be a very practical scheduling tool in a highly dynamic and stochastic environment. Even though the performance of the schedule can be inversely affected by the extra amount of inserted idle times and myopic characteristics of the partial scheduling decisions, the amount of deterioration in the schedule is not always very significant. They also stated that the potential saving in CPU times is great when the partial scheduling can be employed.

Lawrence and Sewell (1997), compared the static and dynamic application of heuristic and optimal solution methods to job shop scheduling problems when processing times are uncertain. They used recently developed optimizing algorithms and several heuristics to evaluate 53 standard job-shop scheduling problems with a makespan objective when job processing times are known with varying degrees of uncertainty. Their results

indicate that fixed optimal sequences derived from deterministic assumptions quickly deteriorate with the introduction of processing time uncertainty when compared with dynamically updated heuristic schedules. As processing time uncertainty grows, they have been able to show that simple dispatch heuristics provide performance comparable or superior to that of algorithmically more sophisticated scheduling policies.

2.2.1.2. Simulation Based: Kutanoglu and Wu (2004), studied methods to improve scheduling robustness under processing time variation for classical job shop problems. They propose a two stage method, actually a combination of static and dynamic scheduling. In this way, they try to achieve scheduling robustness. They first identify decisions that are critical to global performance under the presence of random changes and disturbances. They find and optimize these decisions using a priori stochastic information and Lagrangian relaxation. They call this first stage as the preprocessing stage. This stage is followed by the dynamic adaptation stage, where they apply dynamic scheduling techniques to overcome the effects of changes over time. Kutanoglu and Wu (2004), showed that the developed Preprocess-First-Schedule-Later scheduling scheme is more robust than static optimization schemes while outperforming best known dynamic heuristics in both performance and robustness.

Sabuncuoglu and Kizilisik (2003), worked on a centralized solution methodology for flexible manufacturing systems. They developed a simulation based scheduling system to study reactive scheduling problems in dynamic and stochastic manufacturing environments. They classified the scheduling decisions as *when-to*-schedule and how-to-schedule. When-to-schedule determines the timing between two consecutive scheduling points, and how-to schedule determines the way of generating a feasible schedule. Their simulation results indicate that the variable time response is better than the fixed time-response, and the full scheduling scheme generally performs better than the partial scheduling. They also showed that online scheduling is more robust against the uncertainty and variations in processing times than the optimum seeking offline scheduling.

Kutanoglu and Sabuncuoglu (2001), investigated how simulation-based schemes perform under dynamic and stochastic conditions through an experimental study when simulation is used to identify scheduling policies rather than to generate a complete

schedule. Lawrence and Sewell (1997), showed that the performance of "optimization-based" algorithms used to generate schedules fine-tuned (or even optimal) with respect to deterministic assumptions deteriorate quickly with the introduction of uncertainty. This study questions this case for simulation based methods. The results show that multi pass or iterative algorithm is better than single pass algorithms on average, but not better than the best single pass rule, especially in stochastic cases. This implies that one may just choose the overall best rule and not need to revise this decision for the entire horizon. Additionally the results show that fine tuning a parameter of a rule in a series of iterative simulations may not be a viable approach in a stochastic environment.

2.2.2. Distributed Solutions

Liu and Sycara (1997), focused on the job shop scheduling problem. They present a multi-agent problem solving model and a coordination technique for job shop scheduling. The model involves a group of agents; each agent is associated with either a job or a resource. A solution to a production scheduling problem is the result of coordinated conflict resolution in the iterative and asynchronous multi-agent decision making process. They tried to address the technical need of distributed production management and develop appropriate computational approaches to support adaptive, cost-effective responsiveness. They define the disparity composition ratio, which is the ratio of the number of bottleneck resources to the number of resources in the shop, to divide problems into subsets. In the light of the study of Cheseman (1991), on constraint satisfaction problems, which showed that there is at least an "order parameter" that separates problems in to regions of solvability, they searched for a similar finding for constraint optimization problems. Agents that are assigned to bottleneck resources are more constrained than agents that are assigned to non-bottleneck resources in the negotiation process. By exploiting the special problem structure that involves disparity among agents, they designed coordination strategies based on anchoring on the most constrained agent. Their Coordinated Negotiation Agents (CONA) technique performs well compared to other constraint based heuristic search scheduling techniques. Another finding was a trivial result, but it showed that increasing coordination information decreased total number of cycles, i.e. with more information, the system solves the conflicts more rapidly.

Wang et al. (2003), proposed a heterarchical multi agent system and distributed ruler-based scheduling mechanism. With the distribution of agents and rulers, they were able to divide the scheduling problem into several subproblems, such as decision-making problems for each individual agent, from the agent's own perspective, and coordinating problems between agents for the global goals of the whole system. Rulers are encapsulated in agents; therefore, the complexity of construction, amending, or executing of rulers is reduced and rulers can be reconfigured and reused easily with agents regrouped dynamically. The simulation results have successfully shown that proposed approach is capable of generating feasible schedulers in agile manufacturing environments.

Pendharkar (1999), used a popular dynamic job shop scheduling simulation model that uses distributed genetic learning of job scheduling strategies and study the performance and design issues in multi-agents information systems for dynamic scheduling in manufacturing. Among the design and performance issues considered in this research are, the coordination between agents, number of agents, and frequency of learning. The results indicated that coordination between agents, and learning frequency play a significant role in the performance of multi agent intelligent systems.

2.3. Dynamic Stochastic Problems

2.3.1. Distributed Solutions

Brennan and Norrie (2001), evaluated the impact of dynamic job routing and job sequencing decisions with various job types and loadings, as well as with and without machine failures in their research. They stated that the literature in this area has shown that much of the existing industrial and academic research on manufacturing system control has focused on qualitative comparisons of alternative architectures rather than quantitative comparisons. Their main objective was to compare a heterarchical control (NH) architecture with their proposed unconstrained hierarchical (UH) architecture. Job arrivals and machine breakdowns are stochastic in their test environment. They showed that the proposed UH architecture provided a better flow time when the planning horizon gets longer compared to NH architecture.

Brennan and Norrie (2003), propose two classes of metrics, which can be used to evaluate alternative manufacturing control architectures: those dealing with the controlled system (i.e. the manufacturing system) and those dealing with the control system. In order to illustrate the use of these metrics, the problem of scheduling a simple manufacturing cell is tackled using two test control architectures and the performance of these alternative approaches is evaluated using a modular discrete-event simulation model. Brennan and Norrie (2003), suggest that it is worthwhile to use both manufacturing and control system performance measure to evaluate alternative control architectures, especially when considering dynamic architectures such as a holonic manufacturing system.

Cavalieri et al. (2000), simulated the behaviours of two particular control solutions, characterized by a different degree of decision-making delegation assigned to physical and information units in the production system. The experimental campaign is divided into three trials, in order to measure the performances of the two architectures not only under stable and predictable conditions, but also in case of unexpected events, such as the release of urgent manufacturing orders and the occurrence of failures in production resources. They found out that market-like multi-agent architecture performs well compared to multi-agent architecture with supervisor when the performance is compared against the mean flowtime, whereas the due date performance of the latter is better.

2.4. Review Papers

Baker (1998), made a detailed review of various multi-agent architectures and the claimed advantages for multi-agent heterarchies. Baker (1998), also surveyed the three common types of factory control algorithms: dispatching algorithms, scheduling algorithms, and pull algorithms. He claimed that all the most common algorithms used in industry could be implemented in a multi-agent heterarchy.

Shen et al. (2006a), made a good literature survey on agent-based manufacturing distributed process planning and scheduling. They partitioned their review into four main topics, which are; Approaches to manufacturing process planning, approaches to manufacturing scheduling, approaches to the integration of process planning and

integration, and design and implementation issues. They made an extensive survey on the related topics, and outlined them.

Shen et al. (2006b), made an excellent review about the applications of agent-based systems in intelligent manufacturing. Actually it is an update of their previous review from 1998. They focused on agent based systems for intelligent manufacturing, namely on enterprise integration, enterprise collaboration, manufacturing process planning and scheduling, manufacturing shop floor control, and holonic manufacturing systems. They also studied the literature about the issues implementing agent based manufacturing systems, namely on agent encapsulation, agent organization, agent coordination and negotiation, system dynamics, learning of agents, optimization, security and privacy and tools and standards. They claim that since 1998 there are no significant advancements in this area. It seems that almost all of the research have been directed to the fundamental research to enhance the rationality or intelligence of software agents and develop more efficient and effective coordination and negotiation mechanisms. They also conclude that some difficult problems like full integration of manufacturing process, and control, and particularly integration with real time information from data collection systems remain unsolved.

Vieira et al. (2003), present definitions appropriate for most applications of rescheduling manufacturing systems and describe a framework for understanding rescheduling strategies, policies and methods. This framework is based on a wide variety of experimental and practical approaches that have been described in the rescheduling literature. They also discuss studies that show how rescheduling affects the performance of a amanfacturing system, and conclude with a discussion of how understanding rescheduling can bring closer some aspects of scheduling theory and practice.

2.5. Communication

Many theories of communication exist, but they are essentially based on variations of the theory of communication which emerged from the telecommunications research of Shannon and Weaver (1948). In this model the act of communication consists of the sending of some information from a *sender* to a *receiver(or adressee)*, this information

being encoded with the help of a *language* and decoded upon arrival by the receiver. This information is sent via a *channel* (or medium), which may carry the sound or information. The *context* is the situation in which the interlocutors are placed (this context can itself be broken down into the sender context and the receiver context), as shown in Figure 2.2.

The initial model of this theory was very simplistic and totally bound up with technical matters. Then the concept of communication has become structured and, starting from the initial sending of information, it is moved on to more eloborate forms such as speech acts and conversational structures which place the emphasis on the concept of interaction in communications (Ferber, 1999).

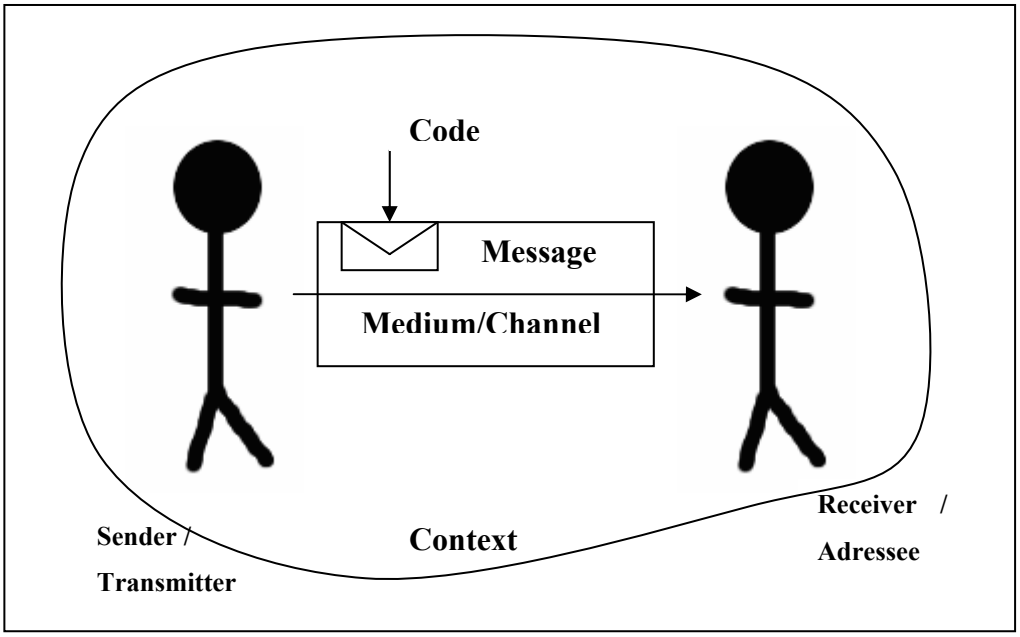

Figure 2.2. Classic model of theory of communication

In multi agent systems, communication is the basis for interactions and coordination. Communication enables the agents to cooperate, coordinate their actions, and carry out predetermined tasks jointly. Several communication languages have been developed for inter-agent communication, and the most widely used ones are KIF (Knowledge Interchange Format) (Genesereth and Fikes, 1992), KQML (Knowledge Query and Manipulation Language) (Finin et al., 1994), and ACL (Agent Communication Language) (Labrou et al., 1999). KQML uses KIF to express the content of a message based on the first-order logic. KIF is a language intended primarily to express the content part of KQML

messages. With the beginning of the standardisation efforts by the Foundation for Intelligent Physical Agents (FIPA), another language called ACL, which is superficially similar to KQML is developed, since than another communication standard has emerged in competition with KQML. After all, XML (Extensible Markup Language), a meta-markup language that provides a simple and very flexible text format for describing structured data, (Goldfarb and Prescod, 2001) and its variants started to become the new standard for agent communication language.

In the literature, blackboard messaging, and message passing are the most studied modes of communication. (Georgeff, 1988; Bourron, 1993; Labidi and Lejouad, 1993; Ferber, 1999; Shen et al., 2001; Weiss, 1999; Wooldridge, 2002, Engelmore and Morgan, 1988).

2.5.1. Blackboard Model

One of the most important precursors to the development of multi agent systems is the blackboard model (Engelmore and Morgan, 1988) The blackboard model proposes that group problem solving proceeds by a group of agents observing a shared data structure known as blackboard, and problem solving proceeds as these knowledge sources contribute partial solutions to the problem. Writing or posting messages to the blackboard is analogous to message sending. Obtaining information from the writings of the blackboard is analogous to receiving messages. Here receiving a message is a Pull type of action, i.e. agent receives the message only when it checks the blackboard, i.e. when it pulls information from the blackboard.

2.5.2. Message Passing

Communication through message passing is a widely used approach. In this approach, agents communicate with each other by sending *asynchronous* messages, while they could also send synchronous messages. In the *synchronous* message case, the sender sends the message to the receiver, but if the receiver is not ready to receive the message the process of sending the message is suspended and queued until the receiver will be able to receive the message. But in the case of asynchronous messaging, the sender sends the

message to the receivers message queue, and does not care whether the receiver could be able to receive it at that instant or not, and the sender can continue its operation. In most agent based applications, the primary mode of interaction is performed through asynchronous communication. Weiss (1999), stated that there are two message types; assertions and queries, which could also be indicated as push and pull. Every agent must have the ability to accept information, actually this condition is called the normal I/O condition by Searle (1969), who extended Austin's (1962), work of speech acts. Formalisms for representing communication in agent theory have tended to be based on speech act theory. Speech act theory treats communication as actions. It is predicated on the assumption that speech actions are performed by agents just like other actions, in the furtherance of their intentions.

In its simplest form, the information is communicated to the agent by means of an assertion. In order to assume a passive role in a dialog, an agent must additionally be able to answer questions, i.e. it must be able to accept a query from another agent and send a reply to the agent by making an assertion. In order to assume an active role in a dialog, an agent must be able to issue queries and make assertions. With these capabilities, the agent then can potentially control another agent by causing it to respond to the query or to accept the information asserted.

Point to point, broadcast, and multicast communications form the most common methods of communication. In point to point (p2p) communication, one agent sends a message to another specific agent, whereas broadcasting enables an agent to send out a message to all other agents in the system. Finally multicasting enables one agent to send out a message to a selected group of agents.

3. BASIC STRUCTURE OF AN OBJECT ORIENTED AGENT BASED SYSTEM

3.1. Structure of the System

In order to be able to work on generic *m* machines, *n* jobs scheduling problems we will need to build a framework, which can represent and simulate the real life shop floor environments. We will try to exploit the advantages of object oriented agent based system structure.

The machines in the shop floor will be represented by the objects. In a formal definition, an *object* is a data holder, it is an organized entity that has attributes and behaviors. An attribute refers to the data encapsulated within the object, whereas a behavior refers to the method used in the system functions.

On the other hand, the failures of machines, or arrival of new jobs, i.e. the realizations of the shop floor, will be represented by the events of the system. *Event* is a special type of a message. In the messaging mechanism, the control agents generate many messages in order to coordinate and control the system. Besides the normal messages, a special type of message will be classified as event, which will cause the system to respond in a predetermined manner. As an example, in the simulation of a breakdown of a real life machine, the simulator agent will fire a breakdown event, i.e. broadcasts a message which states that a breakdown has occurred, and the relevant renewable resource agent catches this message.

The shop floor with all the machines and materials will be simulated with the proposed framework, in order to be able to determine the control and coordination measures of a distributed scheduling environment.

In this distributed scheduling environment there will be decision makers, i.e. agents, which are responsible from a single object or a group of objects. So these agents will coordinate and control the whole system by updating their knowledge of the shop floor,

and by communicating with other agents according to the status of the system, and their decisions.

An *agent* is a special type of *object*, which administer the communication and coordination mechanism. Agents are able to send and receive messages to and from other agents in order to control and coordinate a group of objects. When an agent receives a message, it processes the message by organizing the relevant objects that perform a complementary activity for a common task, and broadcasts the resulting control or coordination order as another message. An object can not communicate directly with another object, every object needs an agent to communicate. Each agent may be related to a specific object, or a group of objects. Agents listen and handle a set of messages which may be related to the group objects which are controlled by the agent, and dispatches a set of messages in response to the received messages.

A *domain* is an area of control or a sphere of knowledge. Agents are responsible for the objects they control, and the agents and the objects together maybe in the same domain. But it is also possible that they could be in different domains. Local domain represents the area of control that is local to the object, i.e. the controlling agent and the controlled objects are in the same domain. Global domain represents the union of all local domains in the system.

The proposed framework has an *agent system*, which is based on object oriented architecture. Each agent is composed of an object. Object orientation supports separation and encapsulation of concerns into distinct objects that overlap in functionality. As an example, procedures, classes, and methods all are defined to encapsulate concerns into single objects, and so the activities of an object are determined. All agents are connected to this agent system.

The agent system operates based on two important mechanisms, the *messaging mechanism*, and the *behaviour mechanism*. The messaging mechanism handles all the communication between the agents, and it enables an agent to send and receive messages. Based on a received message, the receiver agent triggers or activates the controlled objects

behaviour according to the behaviour mechanism. In other words, according to the behaviour mechanism, the objects may perform a state transformation.

Every object has its own attributes, as an example, a job object will have its ready time, processing time on assigned machines, completion time, and other relevant attributes. A set of assignments to these attributes represents a *state* of the object. All possible assignments create all possible states, and form the *state space, Φ.* The state of an object is an instance of the object attributes, i.e. the values assigned to the attributes in an instance defines the state of the object.

The state space can be divided into mutually exclusive subsets according to any criteria that will be meaningful in the context of the relevant object, i.e. according to an aspect of the object. As an example *a named state* called idle can be defined for a renewable resource object, i.e. a machine object, if a machine is not broken and not operating currently, i.e. the state of a machine object of which the TheCurrentBreakdown and CurrentOperation parameters are not set. In that way, a machine can be identified as idle, i.e. in the "idle" named state. So with the help of objects properties and states, one can easily divide any combination of these values into mutually exclusive subsets, and can name them appropriately. To be able to represent the status of an object in a named state from a different aspect, the *substates* are introduced. As an example, one may need to classify a renewable resource object in the "*INUSE*" named state from its speed perspective. In this case, one may define substates like, "fast, normal, and slow", which will help to indicate the status of the working machine more appropriately.

The states, or named states will help us to indicate the status of the objects. As we previously mentioned the behaviour mechanism will perform some state transformations on the objects according to their current states in compliance with the control policy of the system. The behaviour mechanism will need an instruction set, which will indicate what to do in response to certain messages received by the agent. This instruction set is modeled as a visual algorithm and it is called a *state chart*.

A state chart contains a set of named states, events, and actions, and all the relevant relations between named states and events, and between events and actions. The activities

of an object may be separated into distinct aspects, and each aspect can be defined in a state chart. Every state chart defines an aspect of an object, so an object may have many state charts, i.e. aspects that are orthogonal to each other. A state chart acts as an algorithm, or a decision mechanism, actually acts like a map, of what will be done, if the object is in a named state *s*, and an event *e* is realized, and also the new state as a result of these set of transformations that will be executed to the object.

Let

- O is a set of objects.
- E is a set of events.
- Z is a set of actions.
- SC^o_i be the state chart i of object $o \in O$.
- N^o_i: be the named states of state chart i of object o.
- $e \in E$, $z \in Z$

Property 1: At any given time t, an object o may be in only one and only one defined name state in any state chart i.

$$\forall\, x,y \in N^o_i\,,\, x \cap y = \varnothing. \tag{3.1}$$

Property 2: At any given time t, the union of the named states of state chart i of object o, N^o_i, represents the whole state space Φ.

$$\bigcup_{x \in N^o_i} x = \Phi \;\; \forall\, SC^o_i \tag{3.2}$$

As we previously mentioned, a state chart also contains actions. An *action* is a transformation function. It takes an object and its current state, and performs a set of operations, and transforms the object to another state. So with the help of this transformation function, the object makes a state transition.

A *relation* in a state chart indicates a set of pairs between the nodes. In state charts there are state event relations, and event action relations.

A *State Event relation* R^S is a special set of state event pairs. It contains some state event pairs, which are related to the context of the object, and state chart. It may also contain all possible pairs, or only just one pair. A state event relation indicates a set of state event pairs, i.e. { (s_1, e_3) , (s_2, e_1), (s_i, e_j) …}.

` $$R^S = \{(s,e) \mid s \varepsilon N^o{}_i , e \varepsilon E\} \quad (3.3)$$

An *Event Action relation* R^E is a special set of event action pairs. It contains some event action pairs, which are related to the context of the object, and state chart. It may also contain all possible pairs, ore only just one pair. An event action relation indicates a set of event action pairs, i.e, { (e_1, z_4) , (e_e, z_2), (e_i, z_j) … }

$$R^E = \{(e,z) \mid e \varepsilon E, z \varepsilon Z\} \quad (3.4)$$

3.2. Behaviour of the System

Before going in to the details of the behaviour of the system, it would be more appropriate to understand the working of the system as whole. As previously mentioned, the system is an object oriented agent based system to be used in distributed scheduling environments. The Agent System enables the coordination of agents through a communication mechanism which is based on channels, where the messages are sent, or from where the messages are received. In the running of the system events like machine breakdown, or realizations like new job arrival, occurs. As previously defined, each event is actually a special type of a message. There could be many messages sent to an agent, but if the message is related with the agent, there should be a related Message to Event Processor (MEP), in order to handle this message. The agent identifies the context of the message, and finds the appropriate MEP, and if there is no such MEP, then the agent simply ignores the message. The received message will be interpreted by the relevant MEP, and if required a state chart of a relevant object may be activated. Finally at the end of the processing, the resulting decision will be propagated as another message to the relevant agent or agents with the help of the dispatch function. A sample working of an agent can be seen on Figure 3.1.

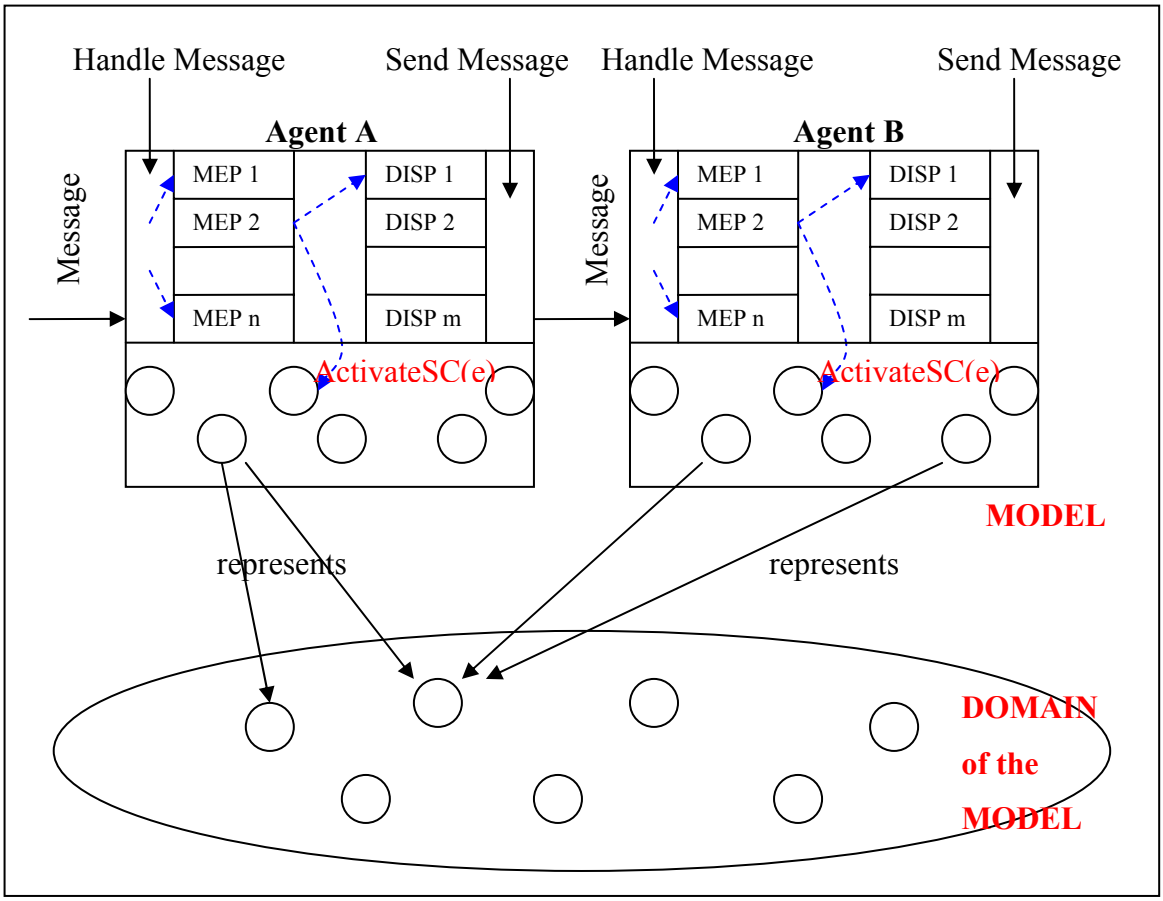

Figure 3.1. Working of an agent

In Figure 3.1 we see two agents, agent A and B, and their internal structure. As it can be seen from the figure the agents have “Handle Message” and “Send Message” functionalities for communication purposes. The circles represent the objects. For example the agent A has a circle with two lines connected the “Real World”, i.e. the domain of the model. It means that the indicated object in the model represents two objects from the real world. This might be the case, where entity 1 and entity 2 in the real world can be represented by an object of agent A in the model, and in this case these entities may be the machines in the shop floor. On the other hand, it might be also the case, where more than one object may represent an object in the domain of the model. As an example Agent B has two objects for representing only one entity from the domain of the model.

When the agent receives a message, the Handle Message functionality handles the incoming message, and redirects it to the appropriate MEP. The Handle Message finds the right MEP by evaluating the incoming message. In the case of Figure 3.1, the MEP2 of

agent A invokes the state chart of an object with an event *e*. Then it dispatches the message with the help of DISP1. DISP1 passes the message to the "Send Message" with the receiver information. Finally the message will be sent with "Send Message".

In the case of a *proxy agent*, there are no MEP or Dispatch functions in the structure of the agent as can be seen in Figure 3.2. It only holds a reference to the remote address of the agent from another domain. More detailed information about proxy agents can be found in Section 3.3.

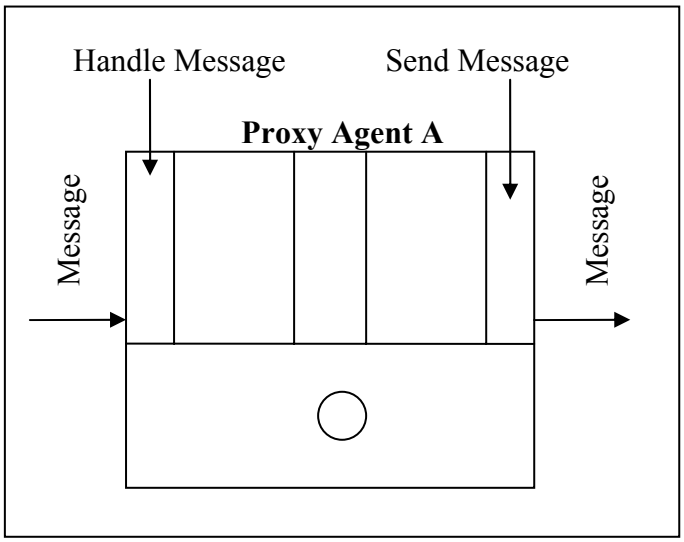

Figure 3.2. Working of a Proxy Agent

The behaviour mechanism determines the decisions of an agent which controls and coordinates a group of objects, and in that way the behaviour, or response of the system, to a series of events and realizations in general. Based on the received message, the receiver agent triggers or activates the controlled objects behaviour according to this mechanism. The behaviour mechanism consists from Message to Event Processor (MEP), Dispatch, Activate Statechart (ActivateSC) and Action.

3.2.1. Message to Event Processor (MEP)

In order to process a message from an agent the Message to Event Processors are used. Agents listen to their registered channels, and on the arrival of a message to their

channel, they first identify the message, and if the context of the message is relevant to the listener agent then the message is processed. The agent may or may not respond to a message depending on the context of the message. If there is no MEP defined for this message, then it simply ignores the received message.

MEP is a handler of system messages, and system messages are generally events like breakdown, or new job arrival. An event is an interpretation of a received message in the context of a state chart. For instance, “breakdown of a machine” as a system message is related to the broken machine itself and also to the current operation on it. The message is interpreted as “breakdown” event for the machine object and as “operation stop” event for the operation object which is currently being processed on the broken machine. These events in corresponding state charts trigger the relevant objects to act depending on the current state of the object.

MEP’s are like functions, and have a sequence of actions to be accomplished. So by the execution of the MEP function, it may trigger or activate a state chart, and/or pass the message to another object or function with the help of the *Dispatch* function.

MEP defines the logical sequences of activation of various state charts of various objects, and dispatching various messages to relevant objects. For instance, again suppose that a machine is broken while it is processing a task, then two state charts will be activated, i.e. the state chart of the machine and the state chart of the current operation on that machine. The process can be designed such that, first the machine changes its state and corresponding event is evaluated, then the events related to the current operation are handled on the relevant state chart. Thus MEP manages the operation of the state charts by providing a logical sequence of activations.

By using the proposed mechanism, messages are allowed to be reinterpreted in different aspects as corresponding events, thus the dependencies between state charts are reduced. Besides MEP can provide event ordering by determining the sequence of the activities of objects it manages, in response to messages received.

3.2.2. Dispatch Function

In order to send a message to a receiver, i.e. to an agent, dispatch function is used. It can also be interpreted as the propagation of the decision made by MEP's to the relevant agents in order to control and coordinate the system. The MEP's define the behaviour of the system according to some events and realizations in the system. In MEP's, the receiver agent decides on certain actions with the help of state charts, and in order to deploy the decisions undertaken, it needs a dispatching mechanism. The message is formed by the agent in a MEP without considering the respondents of the prepared message. The message then will be passed to the dispatch function, and the dispatch function determines the receivers of the message. The composition and the distribution of the messages are isolated from each other, to form two distinct processes.

3.2.3. Activate Statechart function: ActivateSC

The activation of a state chart can be described by a special function, *ActivateSC*, which takes an object *o*, an event *e*, and the time *t* and activates the state chart as an algorithm, *ActivateSC* (o, SC^o_i , e, t), with the following inputs;

- where $o \in O$: the object o of set of objects O.
- SC^o_i : be the state chart i of object o
- e : the event e
- t : time index

A state chart of an object contains the possible named states of the object, the events. This object may need to respond in the form of actions, and possible named state changes as a result of these actions. When the state chart is activated with an event by calling the ActivateSC function, the object will undergo a proper transformation function, according to the current named state of the object, and to the event. The state charts are visual algorithms formed by named states, events, actions, and the relations between the named states and events, and the relations between event and actions.

3.2.3.1. Algorithm: ActivateSC (o, SCoi , e, t): Note that $S_t^o(\omega, c)$ denote the state of object *o* at time *t* according to the global realization, and to the control policy *c*.

Also note that $R^S = \{(s,e) \mid s \in N^o_i , e \in E\}$ represents the state event relations, and $R^E = \{(e,z) \mid e \in E, z \in Z\}$ represents the event action relations

1. Find $s \in N^o_i$, s.t. $S_t^o \not\in s$
2. Find $r^S \in R^S$ where $r^S = (s, e)$,
   if $r^S = \varnothing$ do nothing,
   otherwise continue
3. Let $r^E \subset R^E$ s.t. $r^E = \{(e, z) \mid z \in Z\}$ and $(e,z) \in R^E$
   if $r^E = \varnothing$, END, (*Problem in the State chart, Verification Problem*)
   $\text{Apply}^{\Delta}(z, S_t^o(\omega, c)) = S_{t+\Delta}^o(\omega, c) \in N^o_i$
4. END

Figure 3.3. ActivateSC algorithm

Remarks, and Verification of the State chart: Note that there can be one and only one $s \in N^o_i$, s.t. $S_t^o \not\in s$. If an event has a relation to an action, then there can not be another action which is related to the same event. Therefore the first line following the third statement of the ActivateSC algorithm is possible if and only if there is a problem in the state chart, i.e. a wrong algorithm, or inconsistent modeling.

3.2.4. Action

Action, as the name implies, is actually a transformation function. When applied to an object, the object makes a state transition from its old state to a new state. Actions form an important part of the behaviour mechanism, and they are placed in state charts. So they act as a decision dispatcher, or applier.

When a message is received by the agent, it first finds the appropriate MEP related with this message, and starts to execute it. As we previously mentioned, MEP defines the logical sequences of activation of various state charts of various objects, and dispatching various messages to relevant objects. So when the proper state chart is activated with an event and by calling the ActivateSC function, first the current named state and event relation is found, and after that the ActivateSC finds the proper Event Action relation, so that it is decided to execute the action in this relation. In this way the decision according to the MEP is applied with this action.

3.3. Messaging Mechanism

In distributed systems, groups of similar or complimentary agents act together to solve problems based on their value systems and communication abilities. The objects, in our case agents, communicate with each other with the help of a messaging system. The messaging mechanism is the heart of the overall system, since it enables the agents to coordinate the whole system through communication. Very similar to the Shannon and Weaver's model (1948), the messaging mechanism consists of a sender, a receiver, channels, and a context, but it is an extension of it. In the proposed system, there are channels to which the agents are registered, and with the help of these channels, the agents are able to send and receive messages in an asynchronous manner. The channels of the messaging mechanism actually form Shannon and Weaver's *medium* (1948).

Agents organize objects that own their individual decision making tools and computing facilities specific to their local domains. As an example, from the distributed scheduling perspective, a scheduling problem is decomposed into sub-problems; then the scheduling decisions are shared by appropriate local decision makers, i.e. agents. Agents need to communicate in order to coordinate and control the system. In this distributed scheduling environment, each operation, job, machine or machine group object can be assigned to an agent. Since the agent has two distinct aspects of functioning, the messaging mechanism should be provided to work properly irrelevant to the functioning of an agent. The messaging mechanism of an agent handles the communication issues by registering itself to a channel. Each agent has a receiver to listen to messages and a sender to dispatch resulting messages.

In distributed systems, it may be possible, that the agent and the object to be controlled could be in different domains. As an example, there may be two machines in different shop floors, which are to be controlled by the same control agent. *There should be a mechanism that will resolve the issue of communication transparently, and neither the agent nor the object should care about the insights of this transaction*. To achieve this aim, a new type of agents, *proxy agents*, are created to provide addressing mechanism.

Figure 3.4 shows the functioning mechanism of proxy agents. If an object is controlled by an agent, or if an agent wants to communicate with another agent, and if they are in the same domain, then it is the case of agent A and B shown in Figure 3.4. So there is no need for a proxy agent. However, if the same case occurs for different domains, then there will be a representative of the relevant agent in the other domain.

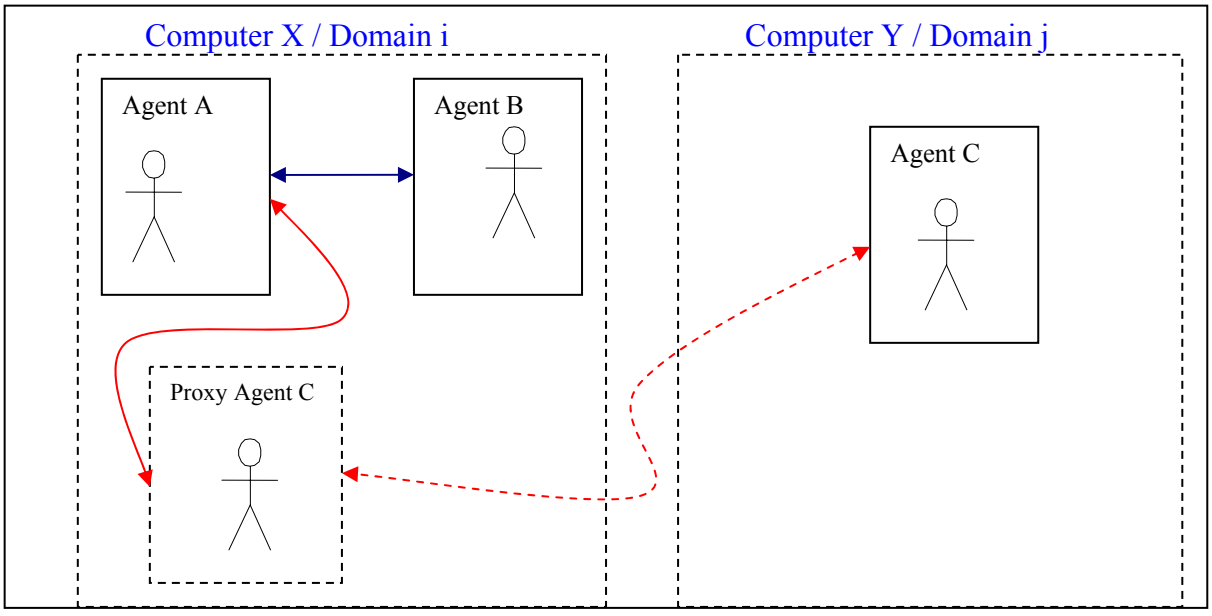

Figure 3.4 The messaging flow among agents

The agent A and agent C are connected through a proxy agent shown in Figure 4.3. When agent A calls for an agent with name agent C, and since the agent C does not locally exist on computer X, i.e. since they are not in the same domain, the proxy agent C receive the message and it redirects the message to agent C which exists on computer Y through a communication media. Actually the proxy agent holds the detailed address information of

the real agent C sitting on computer Y, and re-routes all requests and messages to agent C on computer Y. Each agent has a type and a name, and each agent registers itself to a message channel, with its type and code and holds a local address. The inter-domain communications are provided through these messaging channels. Proxy agents can be thought as the agents that are cloned from the agents in other domains, and they hold the remote address of the agent from the other domain. In other words, if an agent sends a message to another agent, which is not in the same domain of the sender, then the message will be delivered to the addressee through the receivers' proxy agent in the current domain.

4. ISSUES IN DISTRIBUTED SYSTEMS

In order to represent the real world, actually the shop floor data, one needs the help of the objects. In a shop floor, there are many renewable resources, i.e. CNC (Computer Numerical Control) machines, AGV's (Automated Guided Vehicles) etc.. These machines may be in use, idle, or broken and not functioning. These are actually the states of real world machines, which in this study, will be represented with the states of objects. Many unexpected events occur in the shop floor, and the most common events are; machine failure, urgent job arrival, job cancellation, due date change, change in job priority.

In centralized systems, a central decision maker, in this context a central scheduler, knows every bits and bytes of the system, and coordinate and control accordingly. In centralized systems, every action or event is realized in a synchronized manner, i.e. there is a well defined realization and response sequence. But in the case of a distributed system, as the name implies, there is no single entity which can control and coordinate everything, the information flow is performed through communication between the control or decision agents. Asynchronous messaging itself may lead to synchronization problems in the system. As an example, an information about a new job arrival may not reach all the relevant agents on time, and therefore one or more agents may not know the exact system status at that time, so in this case while an agent A knows the incoming of a new job, the agent B may not have any idea about the new job arrival. By the way it could also be possible that Agent A and B may have decided on a common entity two different decisions, which are not consistent with each other.

When you add on top of these, that any decision, knowledge gathering, or implementation process is not instantaneous in real life, and takes time and implement this situation to a distributed system, then the problems will go beyond the previously mentioned ones.

In order to discuss the general distributed scheduling problem, or more generally the issues in distributed systems, a mathematical foundation is needed. As the first step, a mapping between the real life and digital world should be established. While the machines

can be represented by objects, real life events can be represented as special type messages, or events, which are previously defined. An object has a set of attributes, and as it is said before, a set of assignments to these attributes defines a state for the object. The state space can also be divided into mutually exclusive subsets according to any criteria that will be meaningful in the context of the relevant object, i.e. according to an aspect of the object, and these exclusive subsets will be called as named states. The real life states of an object will be represented by states, or named states. In this chapter states and named state terms can be used interchangeably, in order to represent the condition of a real life object, i.e. a machine, at that time.

In this chapter we will try to formalize the mathematical representation of objects, and the common issues in object oriented systems. Then we will go further with the formalization by extending the work to mathematical model, which can also represent a scheduling system.

To be able to model this situation we modified and extended the Strips (Fikes and Nilsson, 1971) model in this study.

Let the following variables to be defined;

- E is a finite set of events.
- S is a finite set of states.
- is a finite set of objects.
- $S_t^o(\omega, c)$ denotes the state of the object o at time t according to the global realization ω, and to the control policy c.

As we previously mentioned, ω represents an elementary outcome, i.e. a realization of the system, which holds all the realizations. This outcome will include all job arrivals, all machine breakdowns, all repair times, all processing times, and all other realizations that may be relevant to the system at a whole. The control policy will be represented with c, which actually controls and determines the behaviour of the system in response to certain events, and also determines how each agent will observe the global realization ω. So c determines the previously mentioned behaviour mechanism. When dealing with the

states of objects under the same realization ω, and the same control policy *c*, we can simply use S_t^o as the state of the object *o* at time *t*. In this chapter when dealing with the state of objects we will simply use this notation.

An object may be in a well defined state, or may be in a situation where a transformation function is being applied to it. When an object is in a stable state it may perform its state-defined actions, and these actions do not cause any change in the definition of its current situation. For example, when a machine processes an operation, it is in "in-use" state. During the time that machine is in in-use state, it performs one of the actions which are specific to in-use state, such as processing a task, and the state of machine is unchanged.

However there are some instances where the object is in a transient and undefined situation, these states are called unstable states. While the object is in unstable state, it may also perform some actions such as actions performed during the state transitions. As an extension of example above, when machine completes an operation, it makes transition from in-use state to idle state. During this state transition, the machine is neither in in-use state nor in idle state, thus the machine is said to be unstable.

The application of some actions on an object, may simultaneously or consequently change the current state of the object, so the object is said to be "unstable" during such applications. In other words, there are some events which may result the application of some transformation functions on the object *o*. These transformations may instantaneously change a set of attributes, so the state of the object may also change. Events may also trigger a transition from state to state and the object performs certain actions as transformations during these transitions. When a transformation function is implemented on an object, then we can say that, the object will be unstable until the execution of this transformation function ends.

Let $APPLY^{\Delta}(\Omega, S_t^o)$ indicate the initiation of an action Ω on object *o* with state S_t^o at time *t*, which is expected to take Δ time units. The APPLY operator takes an action Ω, i.e. a defined transformation function, duration Δ, and a state of the object which will

experience this transformation, and returns a state $S_{t+\Delta}^{o}$. The duration time indicates how long the transformation will last.

$$\text{APPLY: Op x S} \rightarrow \text{S} \tag{4.1}$$

Note: The time of the implementation of APPLY could also be a concern, but throughout this study it is assumed that APPLY operator will be applied to an object, at the time of the current objects state, and so the outcome of the APPLY operator will definitely represent the objects state at time $t + \Delta$..

Let the sign $\rightarrow$ denote the equivalence operator. If $S_t^a \rightarrow S_t^b$, then the state of object *a* at time t represents the state of object *b* at time *t*.

$$\text{Equivalence Operator} \rightarrow : \text{S x S} \rightarrow \text{S} \tag{4.2}$$

Let the *Sync* function takes the states of two or more objects and makes them equivalent according to terms of their domains respectively. In this way the objects on which a sync function is executed will represent equivalent data. As an example if one would like to represent the realization of a due date information of a new arriving job with an object in the scheduling domain, then one may represent it as a job with loose, or tight due date. So there may or may not be a one to one mapping of states in the synchronization of objects.

On the other hand, equivalence of states of two objects are aspect based. Assume that there are two different types of objects, which have some representation of a real life machine, i.e. of the reality in their state definitions. One of the objects represent the real life machine from a scheduling aspect, and say it has named states like, `working`, `broken`, and `idle`. The other object represent this machine from a process control aspect, and say that it is controlling the speed of the spindle of the machine, and has named states like, slow, normal, and fast. So two objects may represent the same machine from different aspects, but they are equivalent to the state of the machine. We can say the two objects' representation of the same machine, *is consistent*.

The representation of data within an object is sometimes required to be updated. For instance to update the object representation of a machine in the shop floor, one has to check the current situation of the machine, since it may have performed unexpected events in the mean time. The updating or synchronizing process is simply reading or gathering information from another data source or object from the same or different domain. To achieve this aim the sync function should be executed.

$$\text{Sync: S x S} \rightarrow \text{S} \tag{4.3}$$

4.1. Simultaneous Transformation Request Problem

Definition: $APPLY^{\Delta}(\Omega, S_t^o) = S_{t+\Delta}^o$. If $\exists T$ such that $S_t^o \neq S_T^o$ for $t \leq T \leq (t+\Delta)$, then object *o* is said to be *unstable* in [t, t+Δ].

Problem: Let the object *o* be unstable in $[t, t+\Delta]$, and $APPLY^{\phi}(\Omega, S_x^o)$. There will be a conflict if $[t, t+\Delta] \cap [x, x+\phi] \neq \{\}$

The problem given above is related to the simultaneous access of multiple processes on the same data encapsulated within the object. This problem will occur when two transformation request to an object is made either concurrently, or just at a time, when the other transformation process is in action.

4.2. Observability Problem

Definition: $APPLY^{\Delta}(sync(a,b), (S_t^a, S_t^b)) = \{ S_{t+\Delta}{}^a, S_{t+\Delta}{}^b\}$. If $S_{t+\Delta}{}^a \rightarrow S_{t+\Delta}{}^b$ then it is *perfect synchronization*, means that the state of object *a* at time $t+\Delta$ represents the state of object *b* at time $t+\Delta$..

Problem: There is no perfect synchronization since the system is not perfectly observable.

This problem occurs because the representation process, i.e. the synchronization process itself takes time. That is taking a snapshot about the current status of the system, and mapping this system to an object can not be performed instantaneously. In distributed object oriented systems there is no single central controller which control or coordinate the whole system, and therefore all objects need to be updated in order to have information about the latest events in the system. As an example, think about the scheduling process of a shop floor. Scheduling a system needs various data about the system, i.e. availability of the machines, list of available operations to be processed etc. In order to get the current situation, data from the shop floor and/or data from an enterprise data source should be gathered. If the scheduling agent starts gathering data at time t, and completes this data gathering process in $t + \Delta$ time units, the main question arises as: "*How accurately does* $S_{t+\Delta}^{o}$ *represent the state of an object o, which may be used to represent a real life machine, at time* $t + \Delta$? Because in the mean time, i.e. between t and $t + \Delta$ this machine may get loaded with another operation from another control agent, or there may be a failure in the machine, and it might break down. It will be also an issue, when two controlling agents, agent a and agent b, try to get an update about the current situation, and started a synchronization process. At the end of the synchronization process, are $S_{t+\Delta}^{a}$ and $S_{t+\Delta}^{b}$ equivalent, or not?

4.3. Conflict of Interest of Agents

Definition: Let $S_t^a \rightarrow S_t^c$, and $S_t^b \rightarrow S_t^c$. If $S_t^a \cap S_t^b = \{\}$ then there exists a conflict.

The above definition states that, if the state of object a at time t represents the state of object c at time t, and if also the state of object b at time t represents the state of object c at time t, and if the state of object a and state of object b at time t have nothing in common, then there exists a conflict. The definition above is related to the system conflicts due to different solution policies among the objects which try to solve the same instance of the problem. Since scheduling problem is not totally separable, there are bounds by some coupling constraints which make independent decisions made by objects to relate to each other. Perfect synchronization is almost impossible to achieve but even it were not the case, i.e. even objects have the same representation of the problem instance, there may be still system conflicts because of the behaviour mechanism of the agents or objects.

That is, for two agents a and b, S_t^a and S_t^b may include decisions which are related to each other through some coupling constraints. The question is, how compatible are these decisions?

Let the operation Ω be a conflict resolution function. *ConfRes* (a,b,c). In order to resolve the conflict resolution function should be defined.

4.4. Robust Representation

Definition: Suppose $S_t^a = S_{t+\Delta}{}^a$, but $S_t^b \neq S_{t+\Delta}{}^b$. If $S_{t+\Delta}{}^a \rightarrow S_{t+\Delta}{}^b$, then representation of state of object b at time $t + \Delta$ by the state of object a at time $t + \Delta$ is a *robust representation.*

In a distributed system with many uncertainties of events and realizations, and with the synchronization issues, robust representation is one of the ultimate goals to be achieved. All possible states of an object should be defined in such a way that the representation of the state remains consistent notwithstanding to the system changes, i.e. it should be a robust representation with respect to system variances.

5. AGENT BASED MODELING OF DYNAMIC SCHEDULING SYSTEM

5.1. Basic Definitions

The scheduling problem will be characterized by an information process, which will be represented by the random variables defined on the probability space Θ. A simulation framework will be set up to analyze the effects of possible real life shop floor events like urgent job arrival, machine breakdown, processing time variability to the system. With the help of this simulation, a realized outcome will be on hand for the system parameter values like; arrival of a job, i.e. ready time of a job, breakdown time of a machine, the repair duration for a machine, and processing time of operations. Throughout this study each and every outcome of these simulations will be defined as a *realization* and will denote it with ω. We will also assume that these realizations will be observed by control entities, which will be responsible from a set of jobs and machines, and these entities, we will call as *agents*. So the agents will have their own observations of the realizations.

To formalize the whole problem we will first let u to represent the problem setting parameters, which include all the distributions for all the random variables, and other things, that may be used in the creation of sample problem sets.

Given the problem setting parameter u, let ω represent an elementary outcome, i.e. a realization of the system, which holds all the realizations. This outcome will include all job arrivals, all machine breakdowns, all repair times, all processing times, and all other realizations that may be relevant to the system at a whole according to the problem setting parameter u.

Let c represents the control policy, which actually controls and determines the behaviour of the system in response to certain events, and also determines how each agent will observe the global realization ω.

Let us first define our fundamental variables;

- J is a set of jobs, M is a set of machines, A is a set of agents
- Q_j is a set of operations of job j ε J.
- Qj^a : all operations of the jobs J^a , which are to be controlled by agent a ε A.
- J^a : a subset of the jobs, which are to be controlled by agent a ε A.
- M^a: a subset of machines, which are to be controlled by agent a ε A.
- t : time index , $t \geq 0$

Let us also define the following random variables;

- $I_j(\omega) = r_j$, j ε J : ready time of job j
- $P_{qm}(\omega) = p_{qm}$, q ε Qj, m ε M : the processing time of operation q on machine m
- $B_{im}(\omega) = b_{im}$, m ε M : the i^{th} breakdown time for machine m
- $D_{im}(\omega) = d_{im}$, m ε M : the repair duration of i^{th} breakdown for machine m

Throughout this study when referring to the realization of the random variable I_j corresponding to ω, we will simply write r_j rather than $I_j(\omega)$. This notation and assumptions will be also valid for p_{qm}, b_{im}, d_{im}.

Let $S_{qt}^a(\omega, c) = \{ F_q , C_q, Y_q , M_q , X_q \}$ denote the state of operation q at time t observed by agent a according to realization ω and control policy c, where

- F_q : start time of operation q
- C_q : completion time of operation q
- Y_q : percent completion of operation q , Y_q ε [0,1]
- M_q : assigned machine of operation q
- Xq : status of being processed or not of operation q, Xq ε $\{0, 1\}$

When we have the states of all operations at time t observed by agent a according to the realization ω and control policy c, the $S_{qt}^a(\omega, c)$ for all q, in our hand, we can easily derive the states of all jobs at time t observed by agent a with the following formula.

Let $S_{jt}^{a}(\omega,c) = \{ S_{qt}^{a}(\omega,c) \mid q \in Qj^{a}, \forall j \in J^{a} \}$ denote the state of job j at time t observed by agent a according to realization ω and c.

Remarks: Here when the operation o is started at time t, then M_q is set as the assigned machine m, F_q is set as t, X_q is set as 1. When the operation is completed than Y_q and C_q values will be set as 1 and t.

Note that $S_{jt}^{a}(\omega,c)$ holds realized values for operations being started, and holds scheduled values for the operations whose ready time is greater or equal than the current time, i.e. $r_j \geq t$. This actually means that $S_{jt}^{a}(\omega,c)$ holds all the realization for all the jobs of which the completion times are less than or equal to the current time t i.e., $C_j \leq t$. It will hold the decision values, i.e. scheduled values, for all the jobs of which the start times are greater or equal to the current time t, i.e. $F_j > t$.

Let $S_{mt}^{a}(\omega,c) = \{ V_m \}$ denote the state of machine $m \in M^{a}$ at time t observed by agent a according to realization ω and c, where $V_m = \{\text{idle, broken, processing}\}$ represents the status of the machine m.

Finally when we have the states of all jobs j, and the states of all the machines m observed by agent a, we can simply derive the state of the system at time t according to the global realization ω, and to the control policy c. Let us define our famous state formula as:

$S_t^{a}(\omega, c) = \{ \{ S_{jt}^{a}(\omega,c) \mid j \in J^{a} \}, \{ S_{mt}^{a}(\omega,c) \mid m \in M^{a} \}$ represents the state of the system as a collection of states of the jobs and machines at time t controlled by agent a according to the realization ω and control function c.

After formally defining the state notations of operations, jobs, and machines through a mathematical notation, the behaviour mechanism, which relies heavily on the messaging mechanism, discussed in the previous sections should be also defined. The Message to Event Processor mechanism and the messages will be integrated to the model with the help of the following notation.

Let $g_{il}^{a}(\omega, c)$ be the time of the i^{th} message of type l received by agent $a \in A$.

Let $l \in L^{a}$ be the set of the types of messages that can be received by agent $a \in A$. With the help of this set l, input messages of a MEP of an agent will be defined.

Finally let $E^{a}_{l}(c)$ be the set of events corresponding to message type l interpreted by agent a based on control scenario c.

It is assumed that a transformation function called $f_e(c)$ is defined in response to event $e \in E_l(c)$. So with the help of this notation the MEP mechanism can be formally defined. When a MEP receives a message of type l at time $g_{il}^{a}(\omega, c)$, it will be interpreted with the help of the set of events $E^{a}_{l}(c)$ corresponding to this message type l, and a proper transformation function $f_e(c)$ will be executed in response to it.

In order to clarify the model let us explain the working of the agent system in terms of our newly developed variables, and operators.

5.2. Working of the Agent System in Terms of the Mathematical Model

Upon receiving a message of type l at time $g_{il}^{a}(\omega, c)$, the agent first identifies the corresponding MEP algorithm related with this message. Then by analyzing this MEP, it also identifies the set E of events that corresponds to this message, and in a pre-determined sequence it executes the proper transformation functions $f_e(c)$ by activating proper state charts and executing the relevant actions $z \in Z$ in them.

When the simulator is initialized it creates the initial simulation events, which are new job event, breakdown event, and a central schedule event. A sample Breakdown event will be analyzed with the help of this notation now. First the initially created or any other new breakdown event during the simulation will be sent to the simulator agent at time t, with a message containing its type and all its relevant content. This case can be formally defined using the above notation in the following way.

Let $s \in A$ be the simulator agent. The message received by simulator agent is a breakdown message; i.e. a message of type 'Breakdown', or with a message code Breakdown which we will discuss in detail in section B.8. Let also L^s = {Breakdown, OperationCompletion, RepairCompletion, CS, Delta_MSG, JobCompleted, NewJob, OperationStart, OperationStop, Register_Delta_Time, SimulationCompleted } be the set of messages that can be received by simulator agent $s \in A$.

The simulator received the breakdown message, which can be indicated with l = {Breakdown}, and $g_{il}^{s}(\omega, c) = t$, be the current time of the i^{th} message of type Breakdown received by agent $s \in A$.

Upon receiving the message type 'Breakdown' at time $t = g_{il}^{s}(\omega, c)$, the simulator agent first finds the relevant Message to Event Processor (MEP), which is in this case, [MEP-Breakdown-OperationCompletion-RepairCompletion] algorithm shown in Figure B.11, and starts to execute it. The MEP will first execute an ActivateSC function with a proper event of the relevant state chart, and then will dispatch the message with the help of the dispatch function.

This MEP contains one ActivateSC algorithm, and one dispatch algorithm. During the execution it first activates the state chart 'SIM_RR', with the event Breakdown, which can be actually represented with the following notation;

$$\textit{ActivateSC}\ (\text{simulator, SIM\_RR, Breakdown, } t) \qquad (5.1)$$

After setting the state of the machine from its current state to the new "broken" state, the MEP continues to execute, and dispatches the same message to their relevant receivers, which are identified in the appropriate Dispatch function, i.e., MACHINE_MSG dispatch function shown in Figure B.12.

The appropriate Dispatch message is activated, and the appropriate function is called, and executed. In this case the simulator agent sends the same message to the Renewable Resource agent which is responsible from this machine.

The same breakdown message is received by the renewable resource agent, and again l = {Breakdown}, and $g_{il}^{a}(\omega, c) = t$, be the current time of the i^{th} message of type Breakdown received by renewable resource agent $a \in A$. This agent receives the same breakdown message after the simulator agent passed it to itself.

In this case, the set of messages that can be received by renewable resource agent $a \in A$ can be denoted with L^{a} = {Breakdown, OperationCompletion, RepairCompletion, Central_Schedule, NewJob, OperationStart, OperationStop, Register_Delta_Time, Release_LS, SimulationCompleted }.

The whole loop will be executed again and again as a standard in agent communication.

6. EXPERIMENTAL SETUP

To test the effects of the various control policy parameters on different problem setting environments we will use a simple single machine scheduling environment. In this setup there is a single machine, which has to process the operations of the jobs currently available, and the ones which will arrive in the near future. The jobs will arrive with a predetermined interarrival distribution, and when they arrive, the main scheduling agent will be aware about this arrival. According to the job system response policy parameter, the incoming job will be either released immediately upon the arrival, or will be released a scheduling interval time later to the shop floor scheduling agent. The control scenario in this case is described in Figure 6.1.

In the single machine case, there are two agents, the first one is the job system scheduling agent, and the second one is the shop floor scheduling agent. The job system scheduling agent acts as a bridge between the shop floor scheduling agent and the real life. All the new arriving jobs will first arrive to the job system scheduling agents, and it will dispatch these jobs according to its control policy parameters. The shop floor scheduling agent is responsible from the shop floor, and this agent is controlling the machine. Since all the job floor scheduling agents have an event based response policy, it handles all the events, and realizations dispatched by the job system scheduling agent regarding to the domain of the machine. In order to see the effects of the decision time, this case with the instantaneous response policy case, and it will be determined as if everything was known a-priori, that is if one knew the release times of the jobs, and solves a static single machine problem with n jobs with ready times known. The static versions of the problems, i.e. the case when all the realizations known a-priori, will be defined as the *utopic problems,* and the solutions to these problems as the *utopic solutions.*

When preemption is allowed, the utopic problem becomes a $1 \mid rj, \text{prmp} \mid L_{max}$ problem, and using the preemptive Earliest Due Date (EDD) algorithm gives the optimal solution. In that way we can analyze the effects of the control policies by comparing the optimal L_{max} value in the static case to the results obtained from the simulation runs.

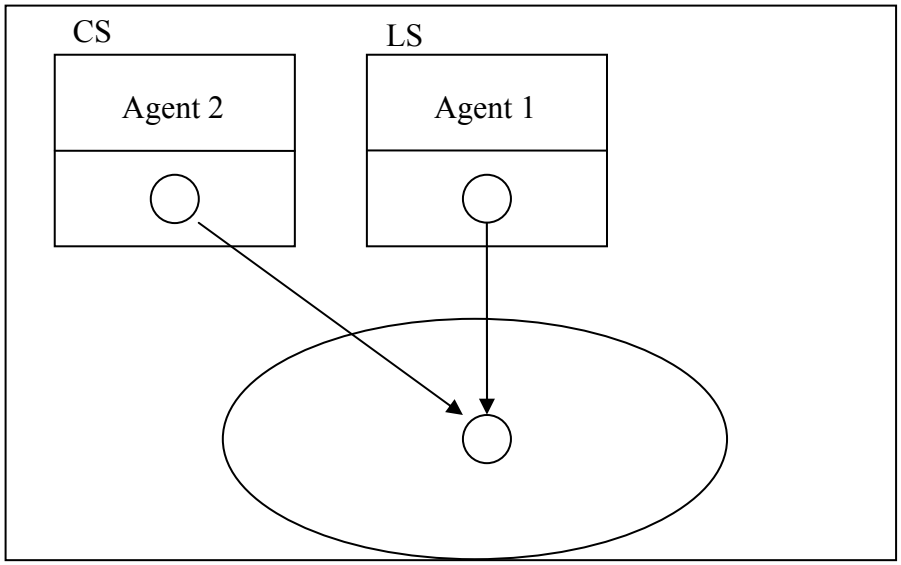

Figure 6.1. Single Machine, Two Agents Control Scenario

In the single machine case, the effects of control policies can be purely analyzed in this setup. The details of, how we will measure the effect of decision time, is explained in section 6.4.4. Also the effects of response policy, agent stability parameters will be analyzed.

6.1. Problem Setting Parameters

In the setup of the simulation system we have a set of control policies and environment variables. The environment variables or the problem settings actually form the problem itself. The problem setting parameter u contains the set of environment variables as previously mentioned in Chapter 5. In the implementation we have the following problem setting parameters in the "Problem Settings" Table of the system database;

- SettingID: The setting ID represents the problem setting ID, so that we can refer to the specified results easily.
- DueDateScaling: The Due Date scaling parameters helps us to adjust the Due Date Tightness of the problem, and actually represents the unitless *parameter k* as discussed in section 6.4.3.
- RealBreakDownDuration: It represents the real breakdown duration in seconds.

- EstimatedBreakDownDuration: It represents the estimated breakdown duration in seconds.
- MinBatchSize: It represents the minimum batch size of an order in units.
- MaxBatchSize: It represents the maximum batch size of an order in units.
- JOBInterarrival: It represents the mean interarrival time of the jobs in seconds.
- InterBreakdownDuration: It represents the mean interarrival time between two consecutive breakdowns in seconds.
- NoOfJobsCompleted: It represents the total number of jobs that should be completed in order on replication of the experiment will be completed.
- CSReschedulingPeriodLength: It represents the rescheduling period length of the main scheduling agent in seconds. It is only used when the job system response policy is cyclic. Note that we will use only multiples of the job interarrival value in this parameter.
- ProcessingVariability: It represents the variability of the processing time. This value is represented as percentage.
- CS_Delta: The decision time it takes for the main scheduling agent during the decision process. It is only used when according to the control policy the decision process takes time for the relevant agent. Note that we will use only multiples of the job interarrival value in this parameter.
- LS_Delta: The decision time it takes for the renewable resource scheduling agent during the decision process. It is only used when according to the control policy the decision process takes time for the relevant agent. Note that we will use only multiples of the job interarrival value in this parameter.
- JS_RP : It stands for the Job System Response policy, it can have the following values, which will be explained in section 6.2; IC, IE, TCS, TCU, TES, and TEU.
- SF_RP : It stands for the Shop Floor Response policy, it can have the following values, which will be explained in section 6.2; IE, TES, and TEU.

The above parameters will be used to adjust the problem settings of the experimental setup. The following parameters will form the environment variables;

- System Utilization = λ / μ

- Due Date Tightness

The system utilization parameter will be calculated by dividing the Job Interarrival rate to Processing rate which is actually the same as dividing the Processing Time to the Job Interarrival Time. Here the Job Interarrival Time is known from the problem settings, and need to calculate the processing time in order to be able to have the system utilization value.

$$Processing\ Time = Batch\ size \times Operation\ Time\ per\ Unit \quad (6.1)$$

where, Operation Time per Unit is set to 60 seconds, i.e. 1 minute in the Alternatives table of the system database for the sake of simplicity.

$$Batch\ Size = Min\ Batch\ size + (Max\ Batch\ size - Min\ Batch\ size) * Random\ Number \quad (6.2)$$

The system utilization parameter will be classified as Low and High, according to their values which are 0.80, and 0.98 respectively. Another important problem setting is the due date tightness. The due date tightness parameter will be classified as tight and loose according to their values which are 5 and 80 respectively. The selection of the values is discussed in section 6.4.

A sample problem setting vector u will denote a problem set with jobs with tight due dates, and high system utilization, and a set of assignments to the values of job interarrival time, breakdown interarrival time and repair duration interarrival times. We will have a total 4 combinations of the environment variables to be tested.

Table 6.1. All four possible combinations of the problem setting parameters

Due Date Tightness	Utilization	Problem Environment
TIGHT	LOW	TL
LOOSE	LOW	LL
TIGHT	HIGH	TH
LOOSE	HIGH	LH

6.2. Control Policy Parameters

The control policies determine the systems behavior when new, expected or unexpected events arrive or happen. All the responses of the system is determined by the control policy setup of the relevant problem. The following items form our main control policies;

- Response Policy
- Decision Process
- Agent Stability during Decision Process

Furthermore there are two layers of control agents in this study, i.e. the main scheduling agent, which is responsible from the whole system, and the local scheduling agents, which is responsible from the machine. The above mentioned control policy parameters will be valid for all of the scheduling related agents. The control policies related with the first one, i.e. the main scheduling agent, will be denoted as the job system control policy, and the other as shop floor control policy.

The response policy could be either cyclic, or event based. In the cyclic response policy case the scheduling agent dispatches a new schedule to the other schedulers at predetermined time intervals, i.e. at each rescheduling interval. This is a valid control policy as a Job System Control Policy, while the same is not true for the shop floor control policy. It is assumed that only the main scheduling agents may work on a cyclic response policy, whereas the shop floor scheduling agent should always work with event based response policy. In the event based response policy case the scheduling agent tries to respond to every predefined event of the system. In our system setup, the predetermined events are breakdown, operation completion, new job, and repair completion events. When any of these events occurs, the scheduling agents with an event based response policy will begin to take actions to respond. When one refers to a response policy the letter C will be used for the cyclic case, and the letter E will be used for the event based case.

In the case of the decision process parameter, it can be either instantaneous, or it may take time. That means any type of the scheduling agents, whether it is a main scheduling

agent or a shop floor scheduling agent, the preparation of the schedule may either takes time, or it may be instantaneous according to the decision process parameter of the relevant control policy. As it is discussed earlier in this study, the decision process is implemented in two steps, the first step is the preparation of the schedule, and the second is the release of the schedule. There are active and passive schedules of a schedule agent, and first the scheduling will be performed on the passive one. In the release schedule step the passive schedule will be assigned as the new active schedule, and the previously active schedule will be the new passive schedule. So if the process takes time the release of the schedule will be postponed a delta time later. When one refers to a decision process one will use the letter I for the instantaneous case, and the letter T for the Takes Time case.

Finally the last control policy parameter is the agent stability during decision process. The agent can be either stable or unstable during the decision process. When the agent is stable during the decision process, it will respond to the messages it receives with the active schedule on its hand during the decision process. The time between the start schedule and release schedule is the time denoted as the decision time, and when one refers to the stability of the agent, one refers the stability of the agent right at this time. So when the agent stability parameter is set to unstable, then the agent will simply ignore any messages it receives during the decision process, and this may lead to some unpredictable results. When we refer to the agent stability we will use the letter S for the stable case, and the letter U for the Unstable case. All the six control policy possibility will be valid for the job system scheduling agents. For the shop floor control policy we only have the Event based response policy, and therefore for a shop floor scheduling agent we will have three different control policies. So this makes a total of 18 (6 x 3) combinations of control policy parameters.

Table 6.2. The control policy parameters at one glance

Decision Process (DP)	Response Policy (RP)	Agent Stability During Decision Process (AS)	Abbreviation
Instantaneous	Cyclic	N/A	IC
Instantaneous	Event Based	N/A	IE
Takes Time	Cyclic	Stable	TCS
Takes Time	Cyclic	Unstable	TCU
Takes Time	Event Based	Stable	TES
Takes Time	Event Based	Unstable	TEU

A set of assignments for these items will form the control policy *c*, which is previously mentioned in chapter 5. A sample control policy *c* would be TCS for the job system control policy, and TES for shop floor control policy. That is the job system scheduling agent will have a cyclic response policy, will be stable during decision process, and its scheduling takes time, whereas the shop floor scheduling agent will have an event based response policy, will be stable during decision process, and its scheduling takes time. These assignments will be written to the problem settings table as the JS_RP and SF_RP fields, which are mentioned in the previous section.

In the case the decision takes time, one will assign proper values for LS_Delta and CS_Delta, i.e. time it takes to schedule for the local schedule agent, and main schedule agent. So the decision time is normalized to the mean job interarrival time, and will use the 0.10 and 0.50 values as the multiplier. That is for both CS and LS delta durations there will be a total of 4 combinations.

6.3. Analyzed Problems

In order to analyze the effects problem setting and control policy parameters on distributed scheduling systems 75 sets of problems will be created according to the environment parameters, and the system will be simulated enough until the number of jobs completed parameter for each problem set, so that one will have simulation results for all the problem sets to be tested. For the problems with decision time a multiplier of two or four whether only one scheduling agent's decision process parameter is "takes time" or both will be there, because of two different LS_Delta and CS_Delta values. Besides these there will be a multiplier of two for the cases when the job system has a cyclic response policy because of two different CSRescheduling values. There are eight control policy cases with both agents having their Decision Process (DP) values as T, and eight other control policy cases with only one agent having its DP value as T and only two control policies have an instantaneous decision process. So by taking all these considerations into account there will be have a total of 300 problem sets to be tested for the system setup. In this setup only two different CSRescheduling values, and two different CS and LS delta values. In order to be able to analyze the effects of the effect of the change of these values

another paired set of values for these parameters will be tested. Adding these experiments will lead a total of 800 problem sets.

Each and every outcome of the simulations performed by the system with these environment parameters and control policies will be a realization ω for the system. One will have a set of realizations given to the selected control policy, and to the selected set of environment parameters.

In general, the makespan C_{max}, total completion time, maximum lateness L_{max}, and the number of tardy jobs can be seen as the most common performance measures in the literature. In this study the makespan, and the total completion time will not be interesting parameters to test because of the infinite number of jobs that has been assumed in the problem setup. Therefore the due date related objectives will be more appropriate performance measures for the current system setup. For these reasons the average lateness L_{mean} will be used as the general performance parameters.

The effects of control policies on single machine will be analyzed. With the help of the single machine problem setup the pure decision time effect and the observability problem will be analyzed.

Although the single machine problem can be seen as simple, even the static version of the simplest case of the single machine problem without preemption is $1 \mid r_j \mid L_{max}$ problem, and it is strongly NP-Hard. In the "Single Machine" problem, the effects of decision time and response policy will be analyzed.

6.4. Random Problem Generation

In order to identify and analyze the effects of the system parameters to the whole system, first the system parameters should be defined appropriately. In this study the following points form the key issues, in the parameter definition stage.

- Job interarrival
- Machine Breakdown

- Tightness of Due Dates
- Decision Time
- Response Policy
- Agent Stability During Decision

The job interarrival time is one of the most important element of the problem setting parameter. All other parameters like processing time, breakdown time, repair time, rescheduling period, scheduling agents decision time, lateness of a job etc., i.e. all time related parameters can be expressed in terms of job interarrival time. So in this way all different problem settings will easily be normalized according to the job interarrival time value.

6.4.1. System Setup

The jobs have their release time r_i, due date d_i, and processing time p_i. The jobs will arrive to the shop floor with an expected job interarrival rate λ. The machines may break down with a probability of γ and will be repaired in β time units according to their repair time distribution. Theoretically there will be unlimited number of jobs, and unlimited number of breakdowns. So a realistic shop floor environment is simulated.

The simulation runs for 250xAvg (d_i-r_i) time units. The test system parameters are constructed in such a way, so that the system could be problem context independent as much as possible. One of the main parameters which the system relies on will be the processing time, which we denote as p_i. The system utilization parameter can be defined as;

$$\text{System utilization: } \rho = \text{work required / total capacity} \tag{6.3}$$

In the long run, this parameter can also be represented as

$$\rho = \lambda / \mu \tag{6.4}$$

where λ is the expected job interarrival rate and μ is the expected job processing time.

6.4.2. Job Interarrival

Exponential distribution will be used for job interarrivals in the system. (Shafei and Brun 1999, Church and Uzsoy 1992). The job interarrival rate will be denoted as slow, and fast according to the following definitions;

Slow: Mean interarrival rate / mean processing time=0.80, system utilization = 0.80
Fast: Mean interarrival rate / mean processing time = 0.98, system utilization = 0.98

6.4.3. Tightness of Due Dates

Baker (1984) defined a set of Due-Date rules. If r_j denotes the arrival time for job j, the we set the jobs due date as

$$d_j = r_j + a_j, \tag{6.5}$$

where $a_j = a_j\ (r_j)$ represents the original flow allowance. The following list describes a number of ways to set the original flow allowances (m_j denotes the number of operations for job j)

CON:	$a_j = k$	(constant flow allowances)	(6.6)
SLK:	$a_j = p_j + k$	(equal slack)	(6.7)
NOP:	$a_j = k * m_j$	(proportional to number of operations)	(6.8)
PPW:	$a_j = p_j + k * m_j$	(processing plus waiting time)	(6.9)
TWK:	$a_j = k*p_j$	(proportional to total work)	(6.10)

The parameter k would be chosen differently for each rule in order to achieve a given average flow allowance.

Total work content rule, denoted as TWK, is one of the most studied due date assignment rule, and this study also uses TWK as the due date assignment policy. Therefore;

$$d_i = r_i + k * p_i \tag{6.11}$$

Baker 1984 and Shafaei and Brun (1999) similarly studied the tightness of due dates with this TWK rule, and they have totally different k values for tight and loose due dates according to their problem context. So the k values have been set as 5 and 80 for tight and loose due dates respectively.

6.4.4. Decision time

One of the main objectives of this study is to analyze the effects of the time it takes to decide on a scheduling, and with the following setup one will be able to analyze the effects.

Let's take a simple single machine scheduling problem, and assume that the system will be rescheduled on every job interarrival. Let's assume that the arrival of the jobs have been simulated, and assume also that the job interarrival pattern is the same as shown in Figure 6.2. If one knew this pattern, then as an example one could calculate the best maximum lateness value for this pattern, if the objective was so.

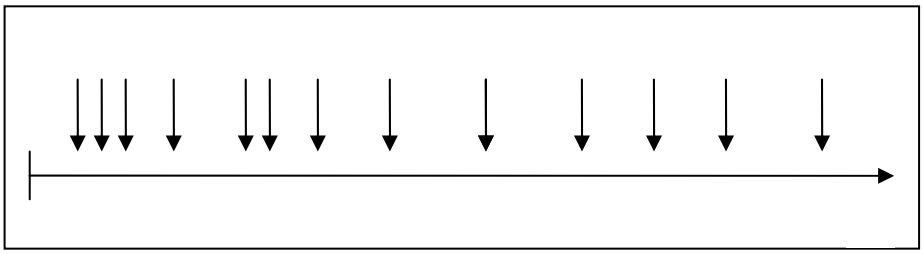

Figure 6.2. Sample arrival of jobs

If the same problem, with exactly the same interarrival pattern, is taken to account, then one can analyze the effect of decision time to the objective. Also assume that the whole problem will be rescheduled upon a new job arrival, but will also simulate the time of the decision process, so that it will take time, and will not be executed immediately. A sample representation of the decision time effect on the system can be seen in the following figure.

Figure 6.3. Decision Time effect on the arriving jobs

6.4.5. Machine Breakdowns

The exponential distribution will be used to simulate the machine breakdowns as it is the case in the job interarrivals. In this study, the busy times approach is used to model machine breakdowns. This method allows the machine to break down when it is busy. A random up time is generated from a busy time distribution and the machine operates until its total accumulated busy time reaches the end of that time. When a failure occurs, a repair time is generated and the machine is kept down during this time period. After that, another up time is generated from the busy time distribution.

To simplify the observability of these distributions we will also define a parameter called efficiency level, which will give the long run ratio of busy time to sum of busy and down time. In this way it will be easier to relate the findings to real world environment.

6.4.6. Response Policy

Cyclic Response Policy: In this policy, the facility will be rescheduled periodically and the resulting schedules will be implemented on a rolling horizon basis. Church and Uzsoy 1992 and others preferred to use absolute rescheduling times, which would make the rescheduling period problem dependent. Therefore a rescheduling interval which is based on the mean job interarrival rate will be used.

Event Driven rescheduling: The system will be rescheduled every time a new job arrives, a machine breaks down, or on an operation completion or repair completion events occur. It might be also a case to implement a cleverer event driven response policy. As an example, such policy might be to reschedule when on a breakdown of a critical machine or

on an arrival of an urgent job. These kinds of events may very well require a significant schedule revision. But for the sake of simplicity and to see the effects of both extremes an event based response policy which will respond to the predefined events in the system will be used.

7. RESULTS

The effects of decision process (DP), response policy (RP), and agent stability during decision process (AS), policies and their respective parameters have been studied in depth with respect to the environment parameters. The instantaneous decision process vs. Takes Time decision process, the cyclic vs. event based response policy, and stable agent vs. unstable agent scenarios will be analyzed. As it is previously defined in the previous sections, there are two scheduling agents, in the single machine case. The first one is the job system scheduling agent, which we interchangeably call as central scheduling (CS) agent, and a shop floor scheduling agent, which we will call as the local scheduling (LS) agent. For the CS agent there are 6 different control policies tested. For the LS agent excluding the cyclic response policy 3 different control policies have been tested, which are tabulated in Table 7.1. So a total combination of 18 different control policies with their respective parameters under 4 different environment parameters has been tested.

Table 7.1. LS and CS agents control policies

Agent Code	SF Policy	Agent Code	JS Policy
LS	IE	CS	IC
LS	TES	CS	IE
LS	TEU	CS	TCS
		CS	TCU
		CS	TES
		CS	TEU

The effect of decision time is analyzed for CS and LS agents. When a CS agent has a decision process which takes time, then in the single machine case, the CS agent will dispatch the arrival of the jobs at the end of the decision process. The agent can be either stable or unstable during the decision process. On the other hand, the agent can have an event based or cyclic response policy. If the agent has an event based response policy, then on every new job arrival, the CS agent will dispatch the arrival of the new job after a decision process. If the decision process is not instantaneous, then it will take time. At the same time, if the agent is stable then it will supply the shop floor, i.e. the LS agent, with the information it currently has. If it is unstable, then it will simply ignore all the requests directed to it, and will only supply information at the end of the decision process.

To clarify, for a TES based JS policy, the decision process will take time, the response policy will be event based, and the agent will be stable during the decision process. On the other hand for a TCU based JS policy, the decision process will take time, the response policy will be cyclic, and the agent will be unstable during the decision process. For the Decision Time parameters the 0.1, 0.25, 0.5 and 0.75 multiples of mean job interarrival time is used. The rescheduling period used in the cyclic response policy has been chosen as the 2.5, 5, 10, and 20 multiples of the mean job interarrival time. To be able to compare the different response policy scenarios, the CS Rescheduling period has been taken as 5 mean job interarrival time, and the relevant delta time has been taken as 0.5 mean job interarrival time.

As it is previously mentioned all the control policies have been tested with 4 different problem setting parameters. These parameters are, Loose Due Date – High Utilization (LH), Loose Due Date – Low Utilization (LL), Tight Due Date – High Utilization (TH), and Tight Due Date – Low Utilization.

To be able to analyze the effects of the decision time from the job system perspective, we will set the shop floor control policy to first as IE, then as TES, and finally as TEU.

In this study, in order to test whether the results differ statistically significant from each other the t-test is used. As the null hypothesis H_0:$\mu_1=\mu_2$ is taken into account, and as the one-sided alternative hypothesis, H_1:$\mu_1<\mu_2$. In this setup H_0 will only be rejected if μ_1 is less than μ_2 and when $t_0 < - t_{\alpha,\upsilon}$, where

$$t_0 = \frac{\bar{y}_1 - \bar{y}_2}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}} \tag{7.1}$$

$$\upsilon = \frac{\left(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2} \right)^2}{\frac{(S_1^2/n_1)^2}{n_1 - 1} + \frac{(S_2^2/n_2)^2}{n_2 - 1}} \tag{7.2}$$

It is assumed that the standard deviations of the compared samples are not equal, which is also verified during the significance tests. For each control policy ten replications of simulations are performed. These values will form the sample sets to be compared. In this study the p-value approach is used. A p-value is the lowest level (of significance) at which the observed value of the test statistic is significant. With the help of this value, the reader can judge the results according to his or her view.

7.1. Effect of Decision Time under IE based SF policy

In order to see the performance of the control policy applied to the system, the difference of the result of the simulated control policy from the utopic solutions Lmean value is calculated. In order to be able to compare this difference result with the results from different problem environments, it is normalized using the job interarrival time. After the normalization every control policy will lead to unitless difference values in every problem environment, which can be easily compared. In Figure 7.1, the difference between the utopic solutions Lmean value and the response policies Lmean value in terms of the mean job interarrival rate is plotted against the different response policies under IE based LS control policy. The CS Rescheduling interval is taken as five, and the CS Delta value as 0.5. To isolate the other effects, the SF policy has been chosen as IE, so the pure effects of the job system agents control policy can be observed. In this figure, it can be seen that the IE-IE scenario leads to the best solution in every problem setting parameter as expected. In this scenario both the CS and LS agents have an instantaneous decision process. So when a job arrives to the job system, i.e. to the CS agent, it will be dispatched immediately to the LS agent, and the LS agent instantaneously reschedules its assigned operations and act accordingly. The IE-IE policy performs better than the TES-IE policy with 0.5 CS Delta with $p=1.28x10^{-10}$.

Here the effects of the problem setting parameter can be observed. In a Loose Due Date-High Utilization environment, the system performance will be the worst compared to the other problem setting parameters, with p=0.0028. One of the main reasons for this to happen is the performance of the utopic solution. In this case since the due dates of the jobs are loose, in the utopic case, the system can find better solutions, compared to the

simulations outcome. The difference decreases as the utilization gets low. In the case of tight due date, the system behaves almost independent from the utilization value, and perform better than the Loose Due Date environments. The system performs the best in the tight due date and loose utilization setup, the values for Figure 7.1 can be seen in Table 7.2.

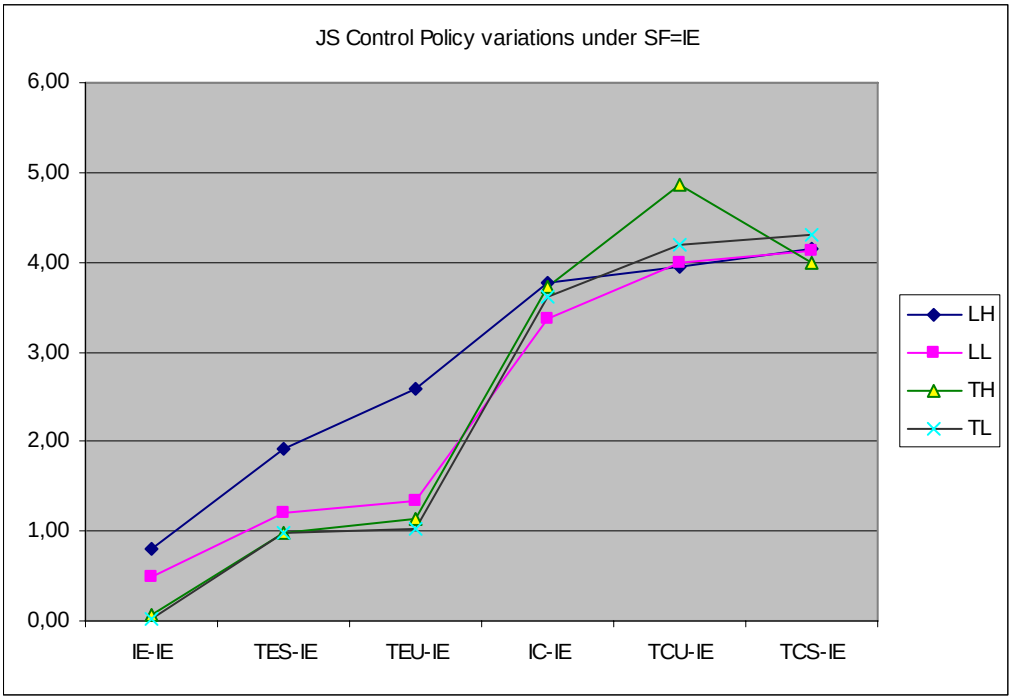

Figure 7.1. Results under different JS Control Policies and IE based SF control policy

The TES-IE scenario leads to the second good solution in every problem setting parameters. In this scenario the CS agent has a TES policy, i.e. an event based response policy with decision process takes time, and a stable agent, and the LS agent has an instantaneous decision process. So when a job arrives to the job system, i.e. to the CS agent, a list of jobs from prior to the new job will be submitted immediately to the LS agent, and after the decision process the agent will dispatch the newly arrived job to the LS agent. Again the LS agent instantaneously reschedules its assigned operations. Again in a Loose Due Date-High Utilization environment, the system performance will be the worst compared to the other problem setting parameters. The difference again decreases as the utilization gets low. In the case of tight due date, the system behaves almost independent from the utilization value, and perform better than the Loose Due Date environments. The

system performs almost the same for the tight due date and loose and high utilization setups with values 0.98 and 0.97 respectively, which can be also seen in Table 7.2.

Table 7.2. The Difference Values in terms of Job Interarrival Times for selected control policies under different problem setting parameters, values for Figure 7.1

	CS Rescheduling Interval	CS_Delta	LS_Delta	L-H	L-L	T-H	T-L
IE-IE				0.80	0.49	0.07	0.01
TES-IE	0	0.5	0	1.92	1.19	0.97	0.98
TEU-IE	0	0.5	0	2.59	1.35	1.15	1.02
IC-IE	5	0	0	3.78	3.36	3.73	3.60
TCU-IE	5	0.5	0	3.94	4.00	4.85	4.19
TCS-IE	5	0.5	0	4.15	4.13	3.98	4.30

The effect of decision time for the TES-IE policy can be seen in Figure 7.2. Except the Loose Due Date – High Utilization problem setting the increase in decision time leads to bigger difference values in all other problem setting environments with p=0.0167. In this LH case the 0.1 multiple of the Job interarrival time as the decision time leads to a bigger difference than compared to the multiple of 0.25, but this difference is not significant, the correspondent p value is 0.1224. From that point on the same behaviour is also analyzed in this scenario. One can not say there is a pattern in the results with respect to the decision time values. But one can say, that an increase in decision time generally leads to more deviation from the utopic solutions. In this control policy scenario, the LS agent is being informed with a lag, but the system informs the shop floor twice in every event cycle, whereas in the TEU case, it informs the shop floor only after the decision process ends.

The TEU-IE scenario leads to the worst solution in event based response polices in every problem setting parameters. In this scenario the CS agent has a TES policy, i.e. an event based response policy with decision process takes time, and an unstable agent, and the LS agent has an instantaneous decision process. So when a job arrives to the job system, i.e. to the CS agent, until the end of the decision process nothing will be submitted to the LS agent. Again the LS agent instantaneously reschedules its assigned operations. In a Loose Due Date-High Utilization environment, the system performance will be the worst compared to the other problem setting parameters. The difference again decreases as the utilization gets low. In the case of tight due date, the system behaves almost independent

from the utilization value, and perform better than the Loose Due Date environments. The system performs the best in the tight due date and loose utilization setup with a difference to the utopic solution as 1.02.

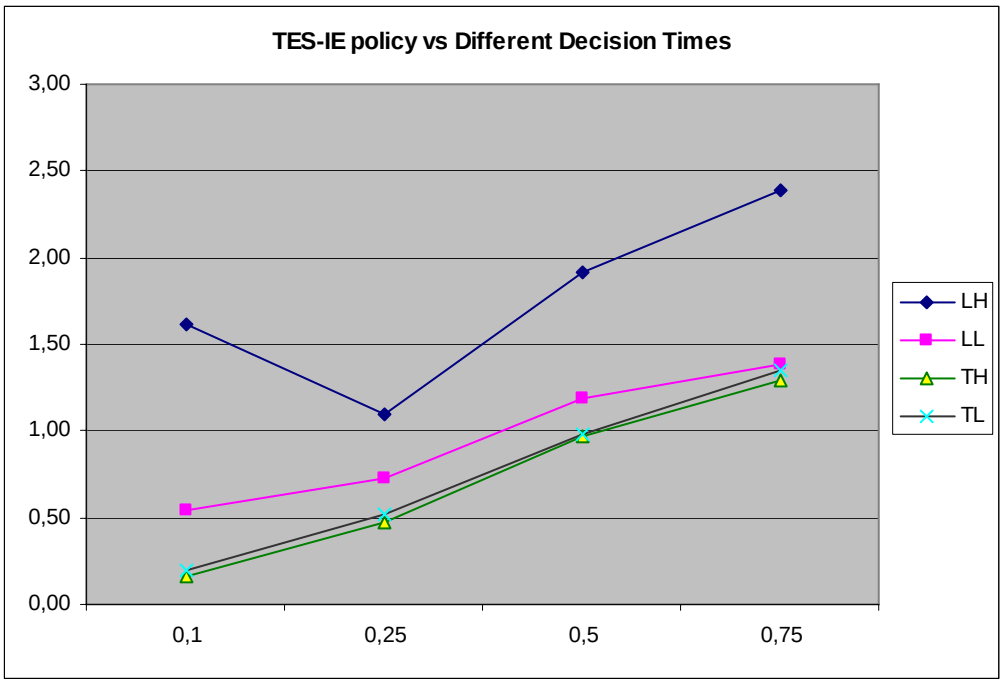

Figure 7.2. The TES-IE policy results vs. Decision Times

The effect of decision time for this TEU-IE policy scenario can be seen in Figure 7.3. In all of the 4 problem setting environments, the increase in decision time leads to poor performance results. In Loose Due Date-High Utilization setup, the system performs the worst. The change of the behaviour of the system can be clearly identified in Figure 7.3 for the remaining 3 problem setting parameters. As the CS Delta value falls in to the second half of the graph the, i.e. for CS Delta values greater than 0.5, the performance of the results of the system gets closer to the utopic values especially for the Tight Due Date-High utilization environment.

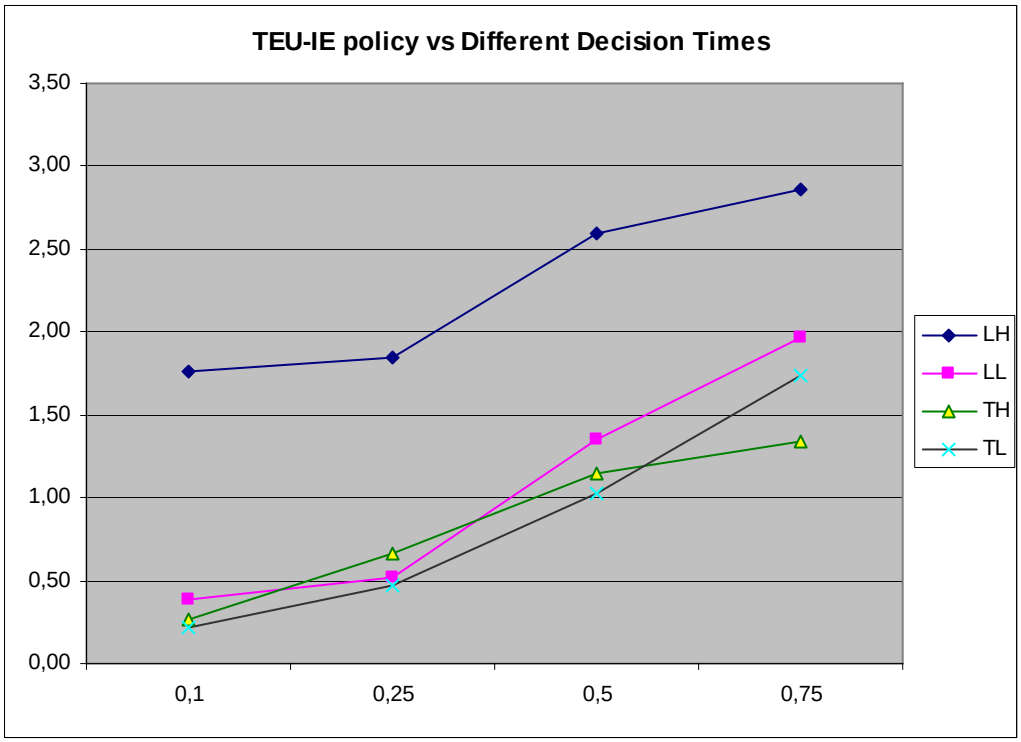

Figure 7.3. The TEU-IE policy results vs. Decision Times

The IE-IE, TES-IE, and TEU-IE scenarios are the top three performing scenarios, which will also indicate the success of the event based response policy in this setup. The first control policy with a cyclic response policy is the IC-IE policy. In this scenario the CS agent has an IC policy, i.e. a cyclic response policy with an instantaneous decision process, and the LS agent has an instantaneous decision process. So when a job arrives to the job system, i.e. to the CS agent, until the end of the rescheduling period, the CS agent will not dispatch the arrived jobs to the LS agent. Only at the end of every cycle the CS agent will dispatch the jobs to the LS agent, and the LS agent will instantaneously reschedule its assigned operations. For this case the high utilization environment almost independent from the due date tightness leads to the worst system performance. The system performs the best in the Loose due date – low utilization scenario, and the second in the tight due date – loose utilization scenario. So in this case the utilization parameter plays the most important role in performance.

Between the cyclic scenarios the TCU performs better than TCS at the selected conditions, i.e. when CS Rescheduling Interval is set to 5 and the CS Delta set to 0.5.

Except the Tight Due Date and High Utilization (TH) case, the TCU policy leads to lower deviations from the utopic solution, which is better. In the TCS scenario, the job system responds at every cycle time with its active schedule first with the information from the previous cycle, and a delta time later, it dispatches the job information up to the beginning of this cycle time. At the beginning of the next cycle time the same information will be resubmitted to the shop floor, which causes a trigger in the LS agent.

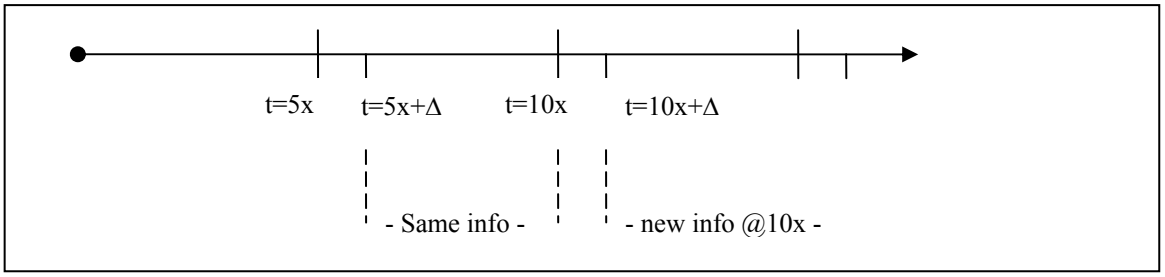

Figure 7.4. TCS control policy at CS agent, and the dispatch intervals, where x represents the mean job interarrival time, and Δ the CS delta time

So the agent stability in cyclic response policy leads to re-dispatching the available information 2 times to the LS agent, which in turn triggers the LS agent to reschedule the assigned operations. So at the beginning of the CS Rescheduling period cycle the CS dispatches the same information a CS delta time later, and at the beginning of the next CS cycle. From the information dispatching perspective the TCU and TCS policy are the same, only the TCS policy triggers the LS agent twice in every CS rescheduling period cycle. When the LS agent gets triggered, it reschedules itself according its SF control policy, which can be IE, TES or TEU, and starts or stops the relevant operations. The LS agent reschedules its operations only when a repair completion event, an operation completion event, or a job system trigger arrives. So in the TCS job system control policy scenario, the job system trigger will reach twice to LS agent compared to the TCU scenario in every CS rescheduling period.

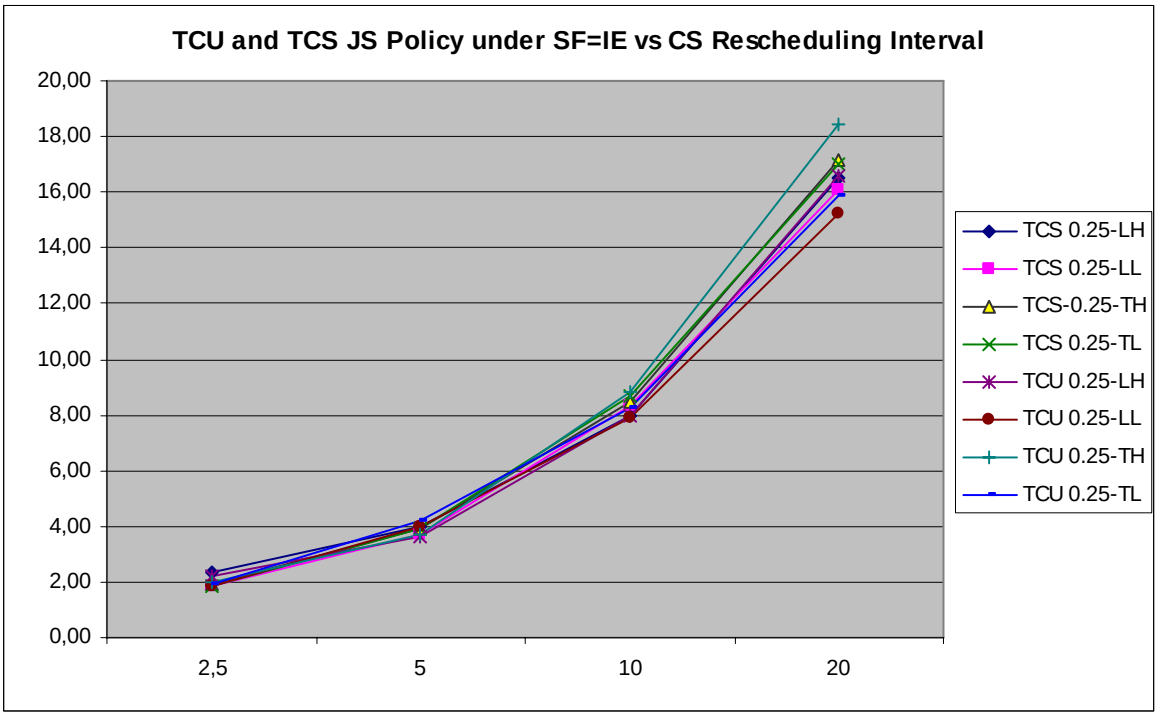

Figure 7.5. TCS vs. TCU JS Policy under IE SF policy under different CSR values

In Figure 7.5, the TCS and TCU job system control policy values are plotted with 0.25 mean job interarrival time as the decision time under the instantaneous and event based shop floor control policy. In this figure one will identify all the differences are very close to each other in every problem setting environment except the Tight Due Date and High Utilization environment for the TCU policy. When the CS rescheduling interval increases, the deviation from the utopic solution also increases. This situation will be analyzed in depth in section 7.5**.**

In Figure 7.6 and Figure 7.7 the variation of the decision time under different CS rescheduling intervals are shown for the TCS-IE and TCU-IE control policies. For both control policy the effect of the decision time is not significant with respect to the effect of the CS rescheduling interval. For every problem setting environment, one can see almost horizontal lines at different levels. The effect of the CS rescheduling interval, in these figures the values for this parameter are 5 and 20, plays the most important role in the differences.

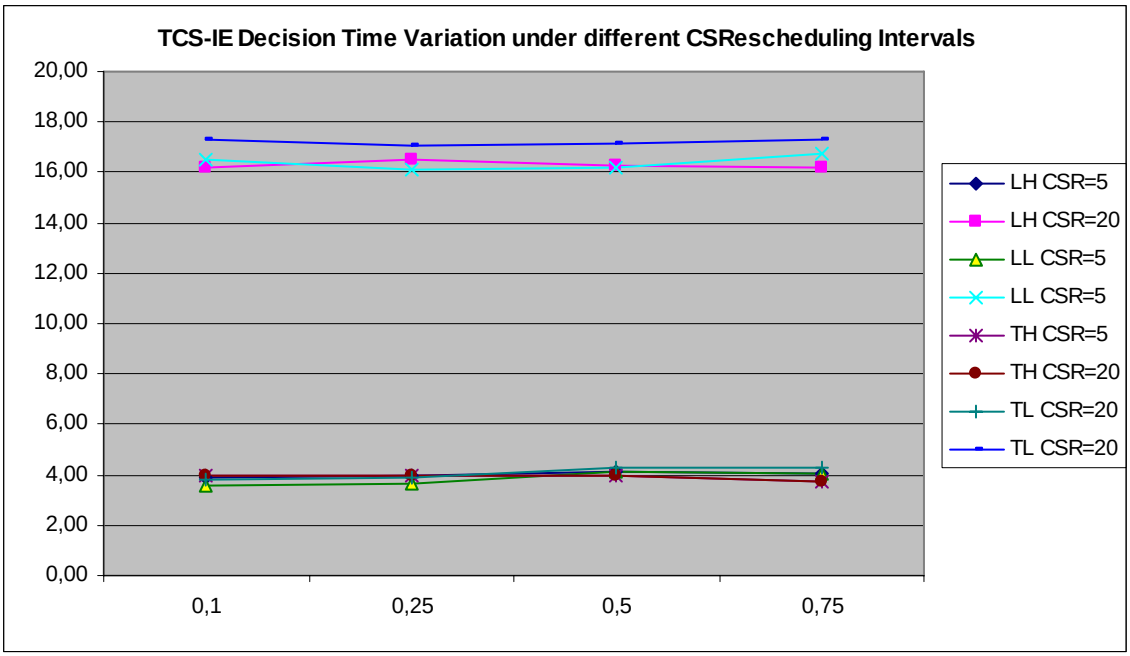

Figure 7.6. TCS-IE Decision Time Variation under different CS Rescheduling intervals

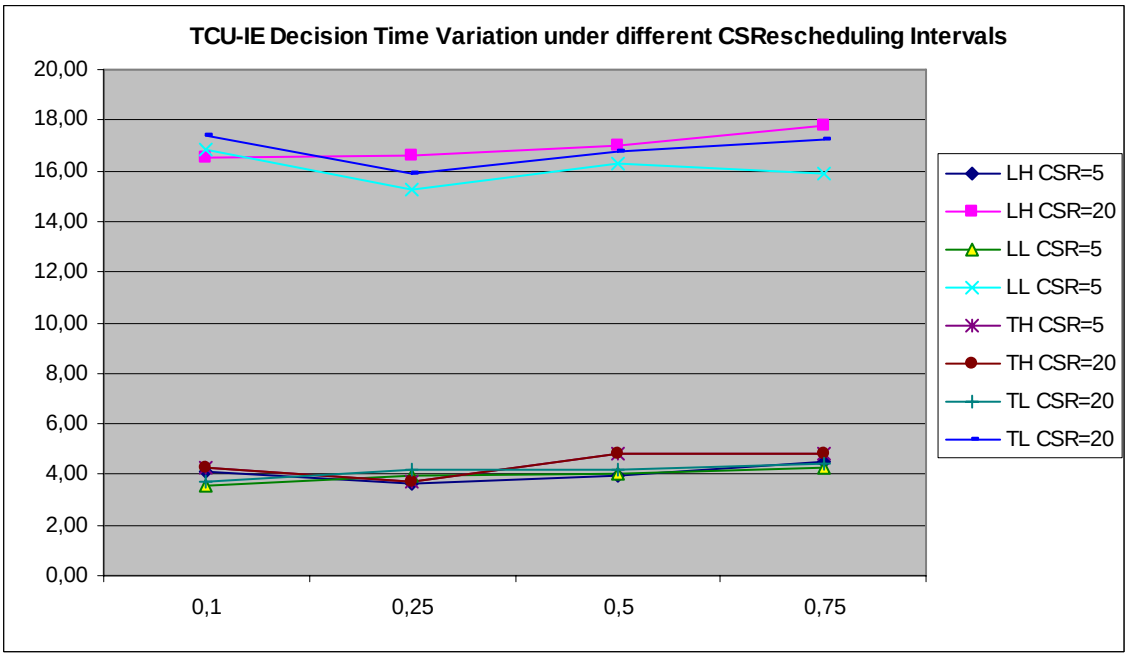

Figure 7.7. TCU-IE Decision Time Variation under different CS Rescheduling intervals

7.2. Effect of Decision Time under IE based JS policy

In this section the effect of decision time is analyzed for the LS agent. The LS agent can either have an IE policy, TES policy or TEU policy, i.e. it has only event based policies. The LS agent will be aware of the jobs only when the CS agent dispatches the relevant jobs to it. When the CS agent dispatches the available jobs according to its control policy, the LS agent will receive the local problem, and will start a decision process according to its own control policy. This decision process can be triggered either via an arrival of an operation completion event, or an arrival of repair completion event, or via the CS agents dispatch. If the LS agents' decision process is not instantaneous, then it will take time, and at the same time, if the agent is stable then it will start or stop operations according to its active schedule, i.e. to the last known good schedule. If it is unstable, then it will simply ignore all the requests directed to it, and will only execute its schedule and start or stop relevant operations at the end of the decision process.

As the scheduling algorithm the LS agent employs the Earliest Due Date (EDD) algorithm in line with the theoretical background of the study. So the operations dispatched by the CS agent will be scheduled by the LS agent according to this algorithm.

To be able to analyze the pure effects of the SF control policy, the job system control policy is selected as an event based instantaneous (IE) control policy. When a job will arrive to the system, it will be immediately dispatched to the shop floor, so the control policy of the LS agent will be the main determining factor in the results. In this case we tested whether the decision time leads to significantly poorer results compared to the instantaneous SF policy, and tested the IE-IE policy against the IE-TES with 0.1 LS Delta value. With even 0.1 LS Delta, the takes time policy performs poorer compared to the instantaneous one with a p value of 0.0004. Since all the IE-TES results with LS Delta values higher than 0.1 are poorer compared to the tested result, we can easily say, it will be true for all IE-TES policies with different LS Delta values higher than 0.1.

In Figure 7.8, the different control policy variations have been plotted under 4 different problem settings. The IE-IE, IE-TES with 0.25 LS Delta, IE-TEU with 0.25 LS Delta, IE-TES and IE-TEU with 0.5 LS Delta control policy results have been shown in

this figure. On top of these control policy setups, another IE-IE policy with random scheduling algorithm is also added to the figure. As expected again, the IE-IE control policy leads to the best results on every 4 different problem settings. The TES policies perform better than the TEU policies under all 4 different problem setting, and in all decision time settings. The selected decision time as the 0.25 multiple of the mean job interarrival time performs better than to the values with the 0.5 multiple. The worst results appear in high utilization problem setting environments.

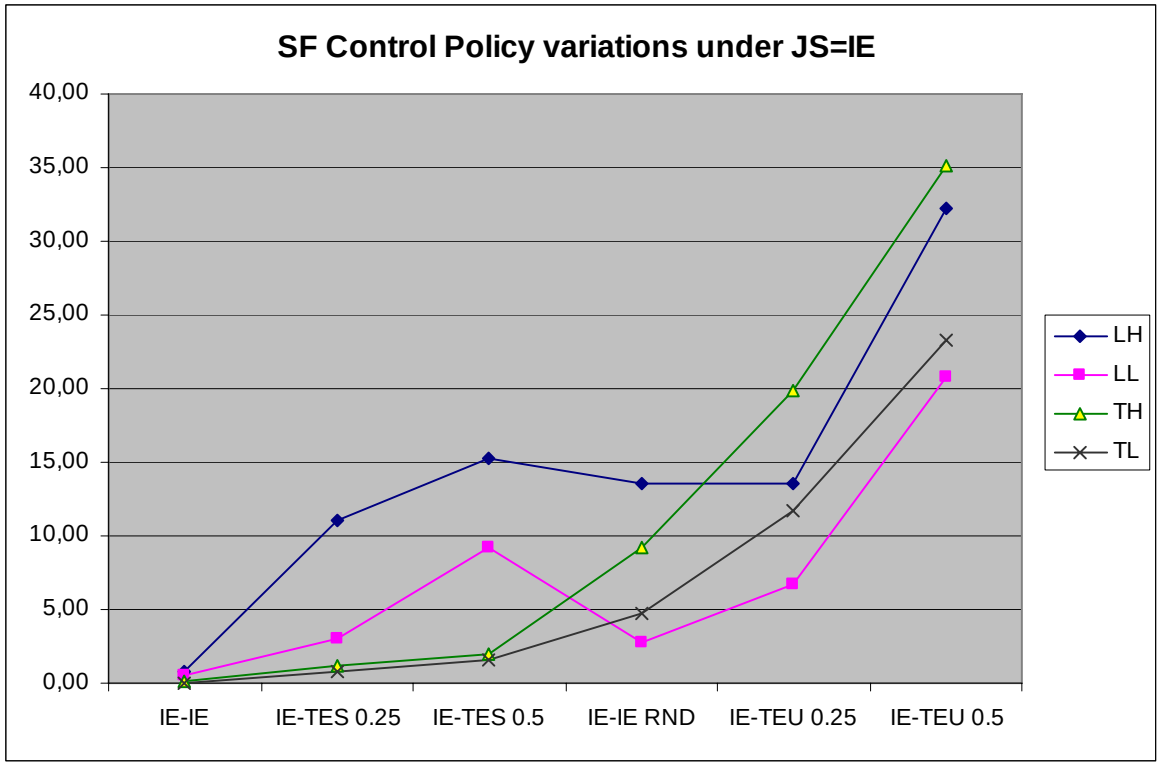

Figure 7.8. SF Control Policy variations under instantaneous and event based job system policy

To be able to analyze the effect of decision time in SF control policy, the results of the IE-TES policy have been plotted against various decision time multiples in Figure 7.9. In the Loose Due Date environments the deviation of the resulting Lmean values from the utopic solutions is higher than the ones with Tight Due Date environments. One of the reasons for this occurrence is the good performance of the preemptive EDD algorithm implemented in the utopic solution. Again one can easily see that an increase in decision time leads to a bigger deviation from the utopic results. In the Tight Due Date

environments the system performs almost the same whether there is high or low utilization of the system.

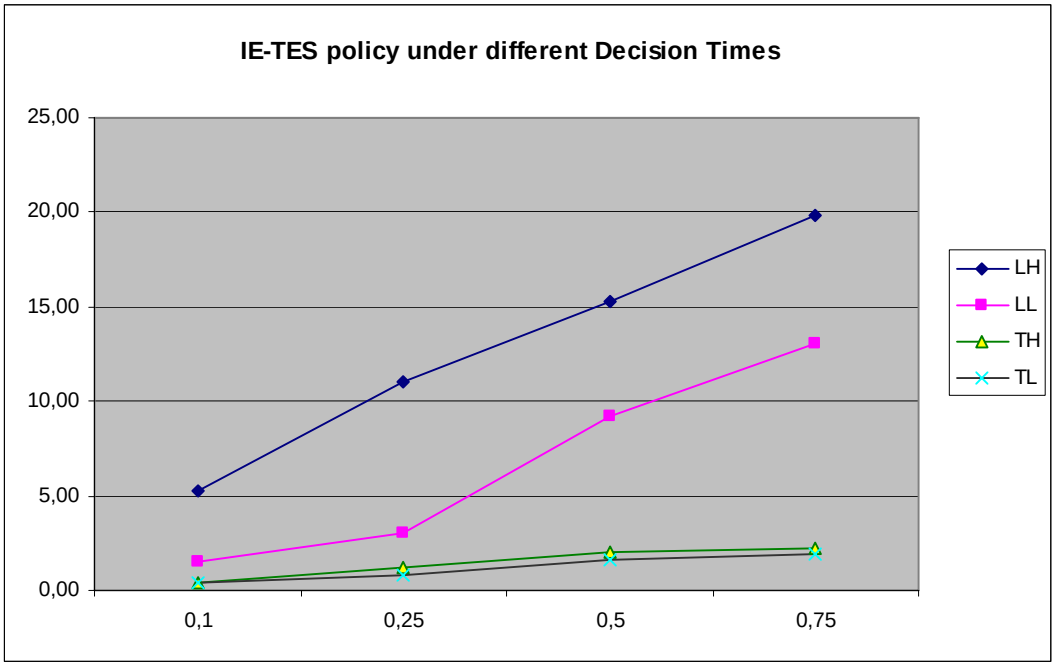

Figure 7.9. IE-TES control policy results vs. Decision Time

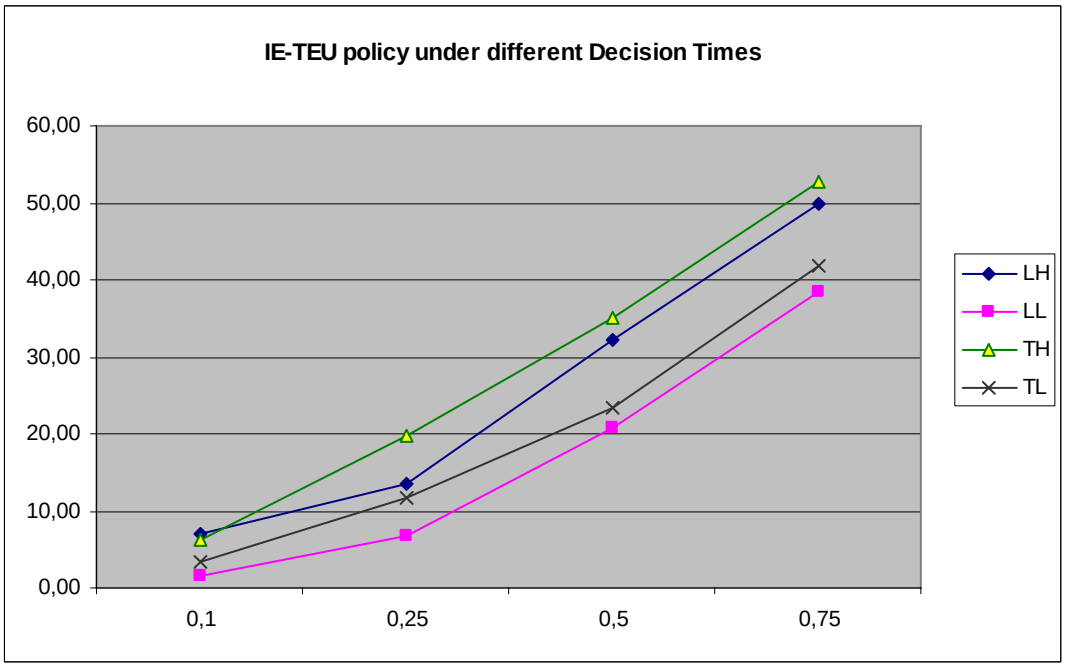

Figure 7.10. IE-TEU control policy results vs. Decision Time

In the case of the IE-TEU control policy scenario shown in Figure 7.10. In this unstable agent scenario, the effects of the decision time can be analyzed more clearly compared to the IE-TES scenario. In this scenario, the high utilization problem setting environments lead to the worst results. In all of the 4 different problem settings, the increase in the decision time, leads to a bigger deviation, and also the patter of this increase is similar in all of these 4 environments.

7.3. Effect of Decision Time under TES based SF Policy

After examining the effect of decision time under IE based SF policy, it is time to analyze the TES and TEU based SF policies. In Figure 7.11 one can see the results regarding to all of the JS control policies under a TES shop floor control policy. In this figure the CS rescheduling interval is taken as the 5 multiples of the mean job interarrival time, for cyclic response control policies. For the decision time duration, i.e. CS Delta, 0.1 and 0.5 multiples of the mean job interarrival is used. All the control policy variations have been plotted against 4 problem setting parameters, namely LH, LL, TH and TL for both CS Delta values. The dashed lines indicate the results for the problem sets with CS Delta value is 0.1.

It is interesting to note that the IE-TES control policy works almost as the worst policy in Loose Due Date environments, the worst for LL and the second worst for LH environment. On contrary to that it performs very well as expected in Tight Due Date environments. For Loose Due Date environments, the IC-TES (IC with CS Rescheduling Interval=5) policy performs as the best control policy. The intuition expected that the event based response policies to perform better than the cyclic response policies. The contrary is true for loose due date environments with a p-value of 0.0001 when the SF policy is TES. For tight due date environments, the expectation realizes and the event based policies perform better with a p-value of 4.62×10^{-9}.

In this scenario the CS agent has a cyclic response policy whereas the LS agent has a Takes Time decision process, event based response policy with a stable agent structure. So when a job arrives to the job system, i.e. to the CS agent, it will be dispatched at every CS rescheduling interval to the LS agent, and the LS agent first responds to the system

according to its currently active schedule, and then after finishing the decision process, reschedules its assigned operations and act accordingly. In Figure 7.11, one can also identify that the cyclic JS response policies significantly perform well compared to the event based response policies when the SF policy is TES under Loose Due Date problem settings. In contrary to that the event based JS response policies perform better than cyclic JS response policies under Tight Due Date problem settings.

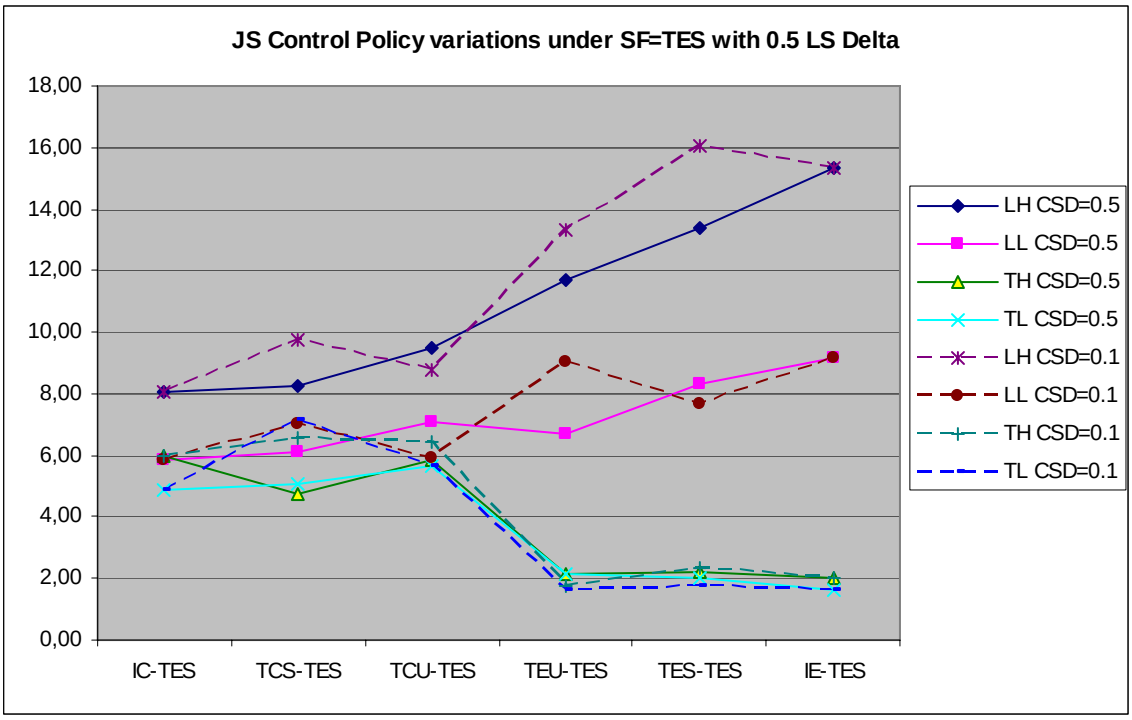

Figure 7.11. JS Control Policy variations under SF=TES with LS Delta=0.5

Another interesting point to note is the deviations of the results in Loose Due Date and High Utilization (LH) environments when the CS Delta value changes. As an example except the TCU-TES policy all the takes time control policies, i.e. TCS-TES, TEU-TES, and TES-TES, perform better when the CS Delta value is 0.5 rather than 0.1. This indicates that when there is a decision time in the CS agents control policy, then delaying the submission of the job information may lead to better overall system performance under LH conditions. A similar situation occurs for the Loose Due Date and Low Utilization (LL) environments. The TCS-TES, and TEU-TES policies perform better when the CS Delta value is 0.5 rather than 0.1.

In tight due date environments the variation in the decision time in the job system level does not lead to a significant impact on the overall results. As expected the IE-TES policy leads to the best results in tight due date environments, and in general the event based JS policies lead to better results compared to the cyclic response policies.

To illustrate the results more clearly the values in Figure 7.11 are plotted in two separate figures shown in Figure 7.12 and Figure 7.13 with respect to the CS Delta values in the job system level. With the help of these two figures, it can easily be seen that for lower CS Delta values the cyclic response policies generate similar results regardless of the decision process (DP) for LL, TH, and TL environments. Again it can be observed that actually mainly for tight due date environments the tested response policies generate similar results.

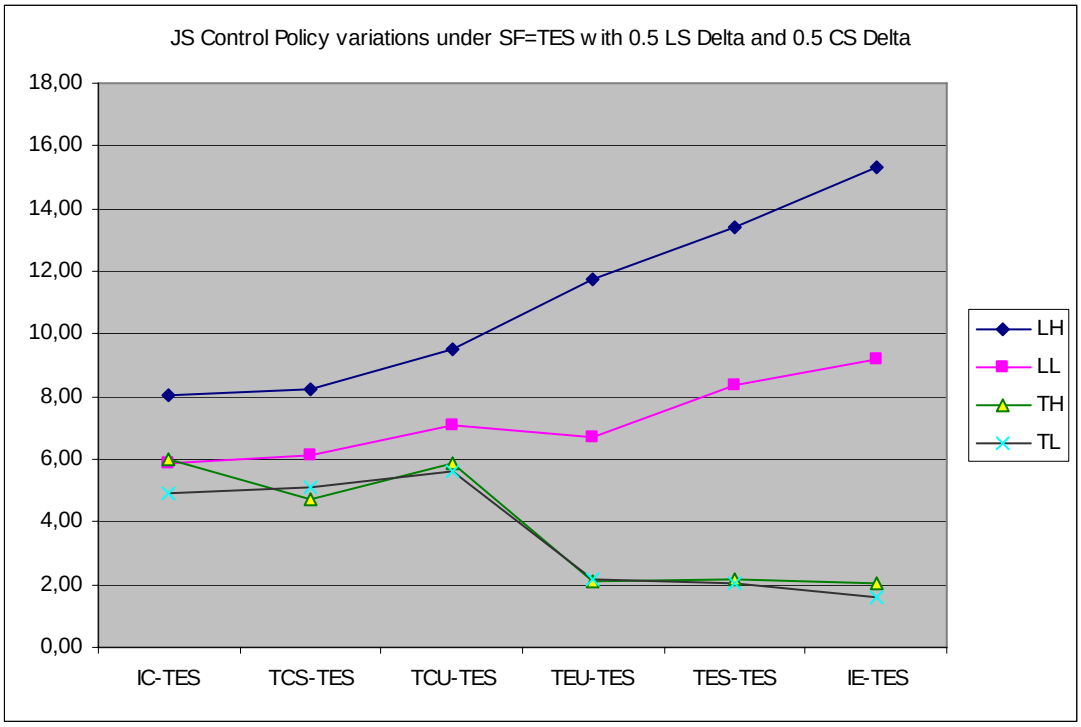

Figure 7.12. JS Control policy variations under SF=TES with CS Delta=0.5 and LS Delta=0.5

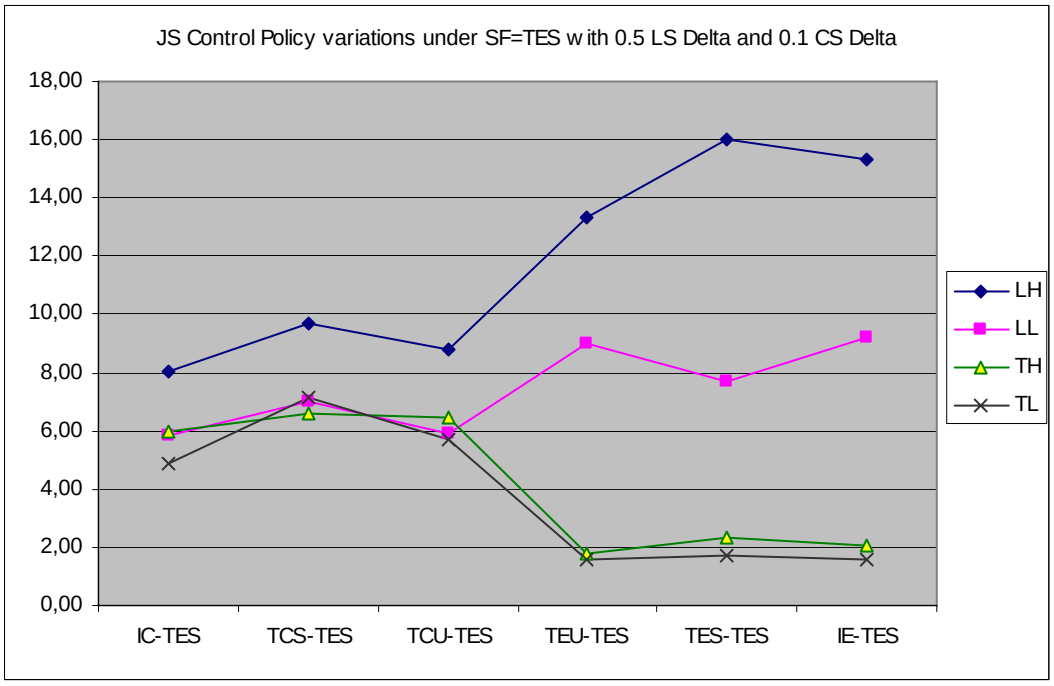

Figure 7.13. JS Control policy variations under SF=TES with CS Delta=0.1 and LS Delta=0.5

The effect of decision time for the TES-TES policy can be seen in Figure 7.14. In this figure the major gridlines on to the x-axis create 4 zones with LS Delta values 0.1, 0.25, 0.5, and 0.75 respectively. The CS delta and LS Delta values are appended together as the x-axis labels, and as an example 0.1-0.5 control policy indicates that there is an event based takes time control policy with 0.1 CS Delta implemented with a stable JS agent, and an event based takes time control policy with 0.5 LS Delta implemented with a stable SF agent. From this figure one can easily observe that the second major gridline acts as a border at LS Delta=0.5. So for LS Delta values less than 0.5, an increase in the CS Delta value leads to higher difference values which means poorer results for all the problem setting environments. But from that point on, i.e. for LS Delta values greater than 0.5, an increase in CS Delta value leads to a decrease or almost no or little change in results. For LH environments delaying the job submission to the shop floor leads to better results for LS Delta values 0.5 and 0.75. The same is valid for LL environments when LS Delta value is 0.75. The Tight due date environments exhibit small changes in increasing decision time values.

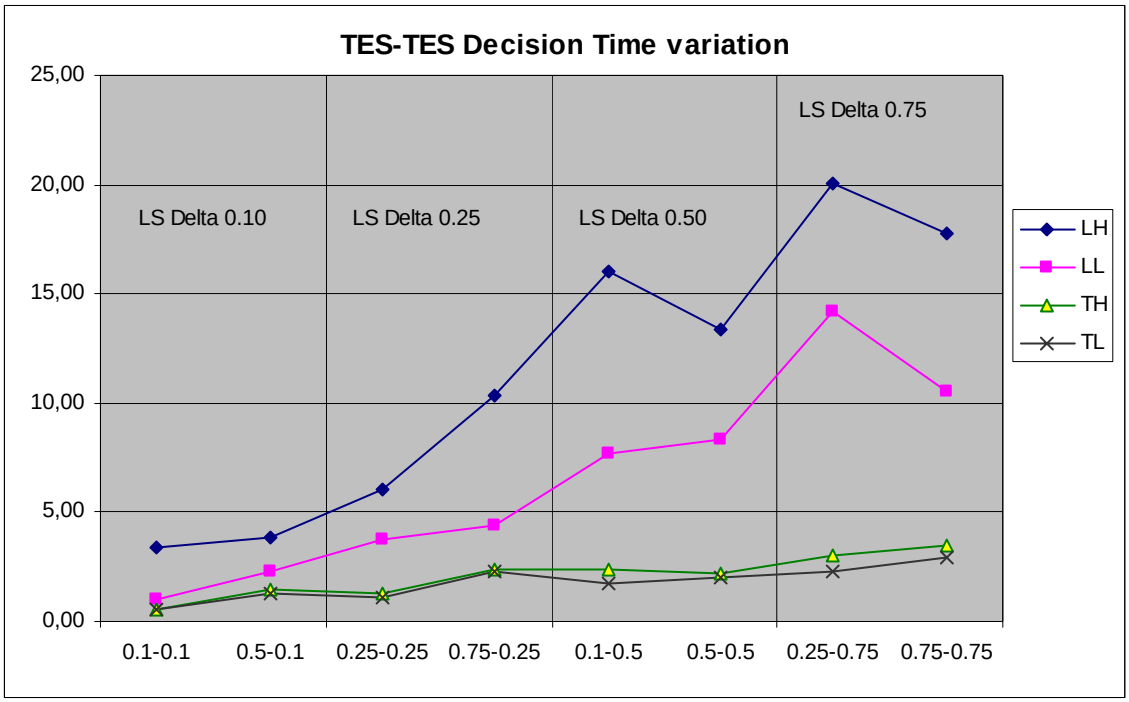

Figure 7.14. Decision Time variations under TES-TES control policy

In Figure 7.15 one can see the effect of decision time under TEU-TES control policy. In this figure again the major gridlines on to the x-axis create 4 zones with LS Delta values 0.1, 0.25, 0.5, and 0.75 respectively. The TES-TES policy with 0.5 CS Delta values performs better than the same policy with 0.1 CS Delta with p=0.0057. An increase in CS Delta value leads to poorer results in every problem setting environments, except the 0.5 LS Delta zone. Only in this zone the Loose Due Date environments exhibit a better performance by an increase in CS Delta. In all other parts of the graph it can be clearly seen that an increase in decision time whether in CS Delta or LS Delta leads to poorer results for the whole system. That means for LH and LL environments delaying the job submission to the shop floor leads to better results when the decision takes time for 0.5 mean job interarrival time for the LS agent. The Tight due date environments exhibit small changes in increasing decision time values as in the case of TES-TES policy.

In Figure 7.16 the effect of decision time variation on TCS-TES control policy is observed. In this figure the major gridlines on to the x-axis create 4 zones with LS Delta values 0.1, 0.25, 0.5, and 0.75 respectively. In this figure again one can easily observe that the second major gridline acts as a border at LS Delta=0.5. So for LS Delta values less than

0.5, an increase in the CS Delta value leads to poorer results for all of the problem setting environments. But from that point on, i.e. for LS Delta values greater than 0.5, an increase in CS Delta value leads to a decrease in all problem setting environments. For LH, LL, TH, and TL environments delaying the job submission to the shop floor leads to better results for LS Delta values 0.5 and 0.75 for TCS-TES policy.

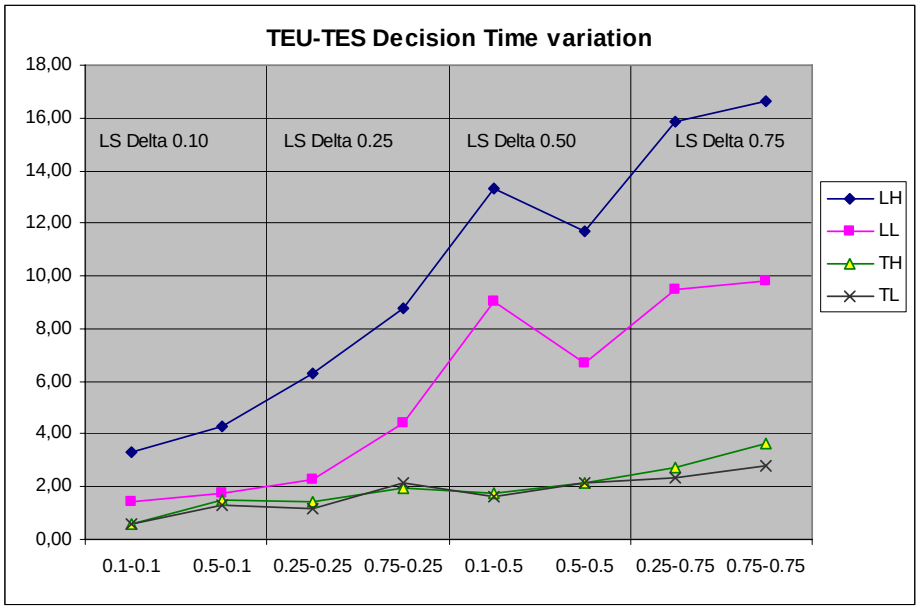

Figure 7.15. Decision Time variations under TEU-TES control policy

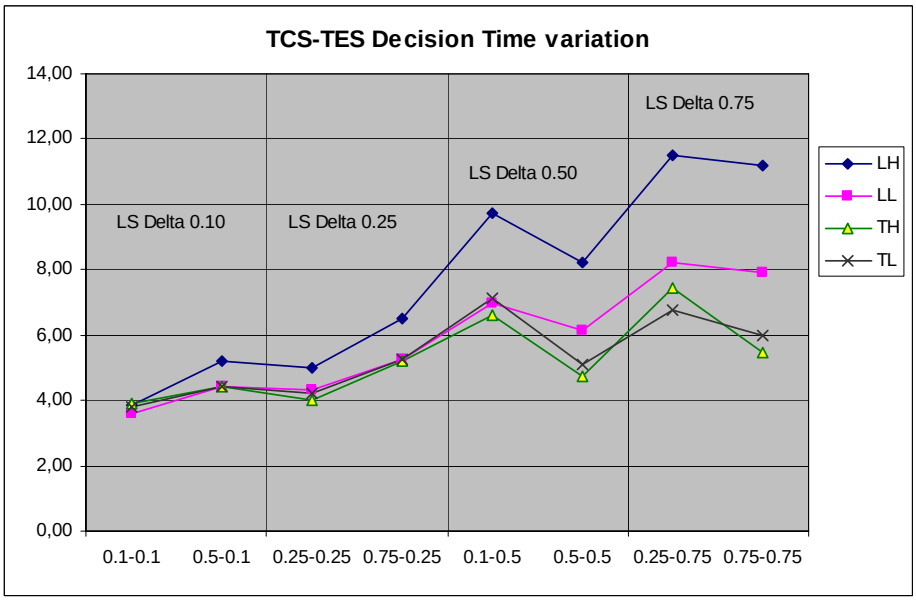

Figure 7.16. Decision Time variations under TCS-TES control policy

In Figure 7.17, the effect of decision time variation on TCU-TES control policy is observed. In this figure the major gridlines on to the x-axis create 4 zones with LS Delta values 0.1, 0.25, 0.5, and 0.75 respectively. In the TCS job system control policy scenario, the job system trigger will reach twice to the LS agent compared to the TCU scenario in every CS rescheduling period. When the SF policy is TES that will lead to differences between TCU-TES and TCS-TES control policies. In this figure again one can see that an increase in decision time in the LS agent leads to poorer results. In this case for LL at LS Delta 0.1, for LH at LS Delta 0.25, and for TH at LS Delta 0.5, an increase in CS Delta from 0.1 to 0.5 leads to better results. In all other cases an increase in CS Delta leads to poorer results. Another thing to note that the LL, TH, and TL results converge to each other at 0.1-0.1 i.e. at the lowest decision times, and at 0.75-0.75, i.e. at the highest decision times.

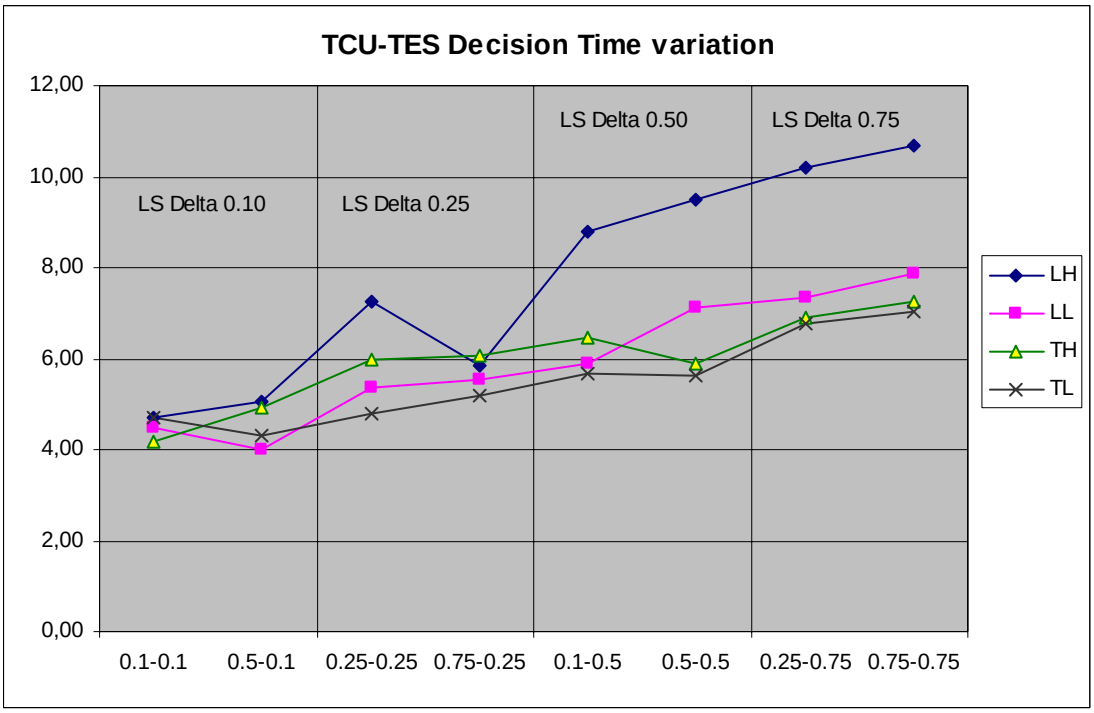

Figure 7.17. Decision Time variations under TCU-TES control policy

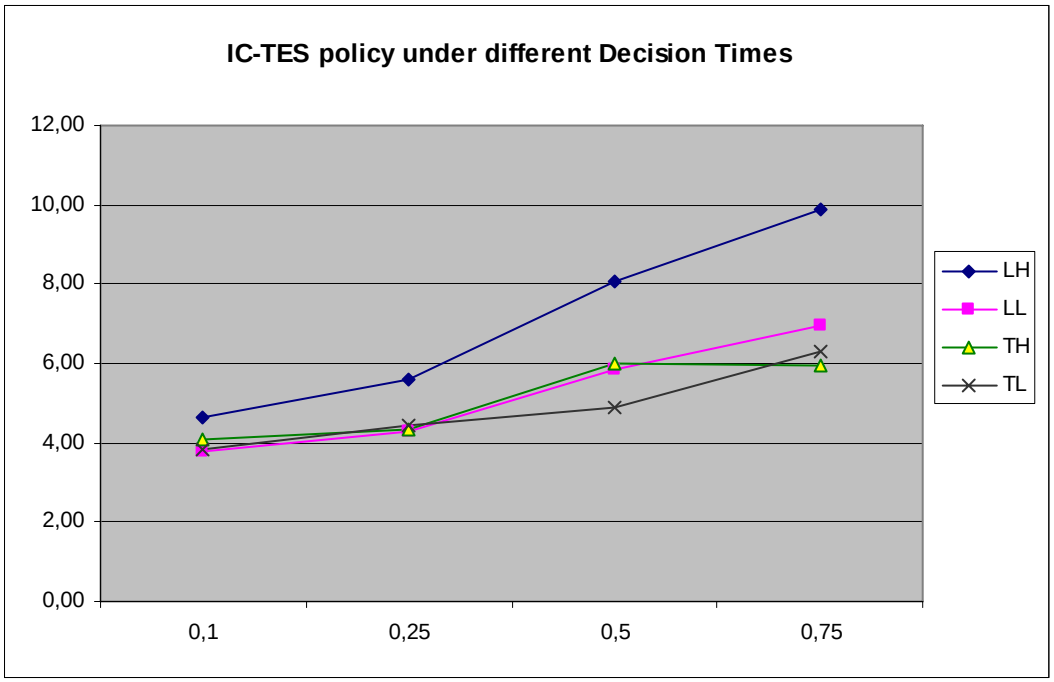

Figure 7.18. Decision Time Variations under IC-TES

In Figure 7.18 the TES SF policy is examined when the JS policy is instantaneous and cyclic. It can be seen that in almost every problem setting environment an increase in decision time leads to higher deviations from the utopic solution. Only in the tight due date high utilization environment while an increase in decision time from 0.1 to 0.5 leads to poorer results, the deviation do not further increase from 0.5 to 0.75 LS Delta values.

7.4. Effect of Decision Time under TEU based SF Policy

In Figure 7.19 and Figure 7.20, the CS rescheduling interval is taken as the 5 multiples of the mean job interarrival time, for the control policies with cyclic response. For the decision time duration 0.1 and 0.5 multiples of the mean job interarrival is used. In Figure 7.19 one can see the results of the JS control policies under a TES shop floor control policy and in Loose Due Date problem settings, whereas the Tight Due Date results are shown in Figure 7.20. The dashed lines indicate that the CS Delta value is 0.1 for both figures. In these figures one can note that the TES-TEU policy performs the best for 0.5 CS Delta values.

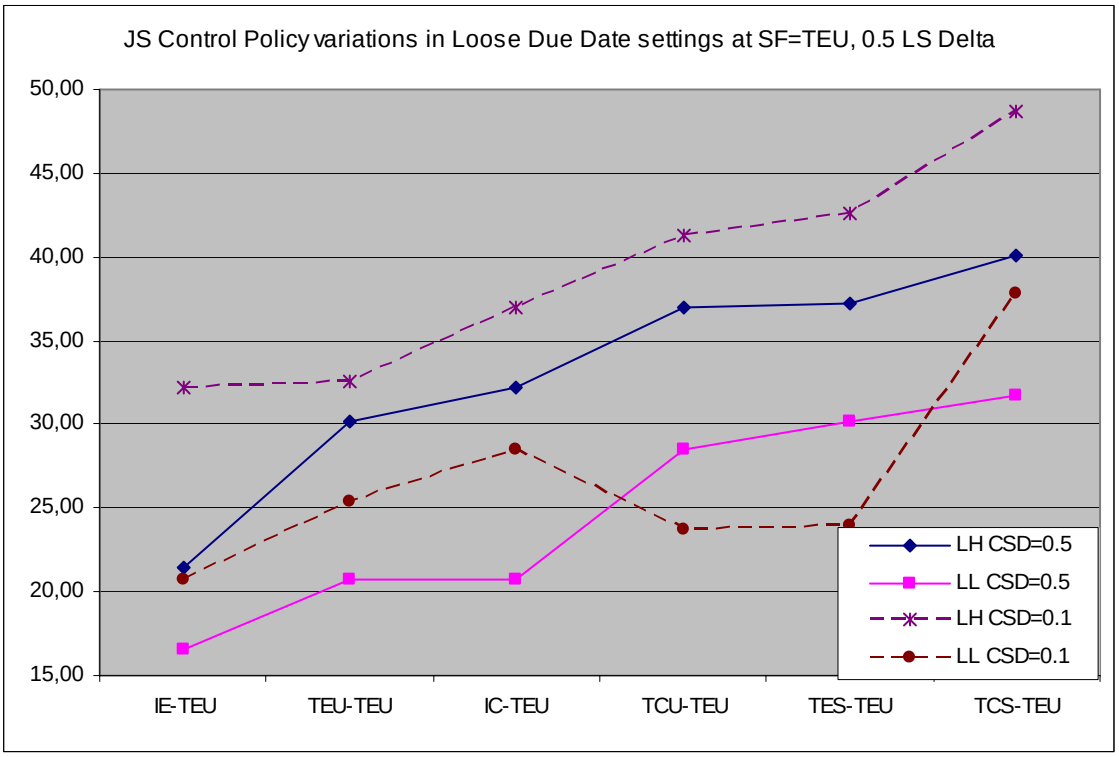

Figure 7.19. JS Control policy variations in Loose Due Date settings at SF=TEU, 0.5 LS Delta

In TES JS policy, the CS agent sends the job information twice compared to instantaneous policies, and also compared to takes time but unstable scenarios. For the High Utilization environments, i.e. for LH and TH environments, an increase in CS Delta leads to better results in both event based and cyclic JS policies. The same is also true for LL environment except the TCU-TEU policy. In tight due date low utilization environments a change in CS Delta does not have a significant effect on the results.

The effect of the JS control policy variations under 0.5 LS Delta and 0.5 CS Delta setting can be seen in Figure 7.21, and the same settings with 0.1 CS Delta is plotted in Figure 7.22. For 0.5 CS Delta value, the TES-TEU is the best, and the TCU-TEU the worst performing policy for all problem setting environments. But for the 0.5 CS Delta value, except the TL environment the IE-TEU policy works the best, and the TCS-TEU is the worst performing policy.

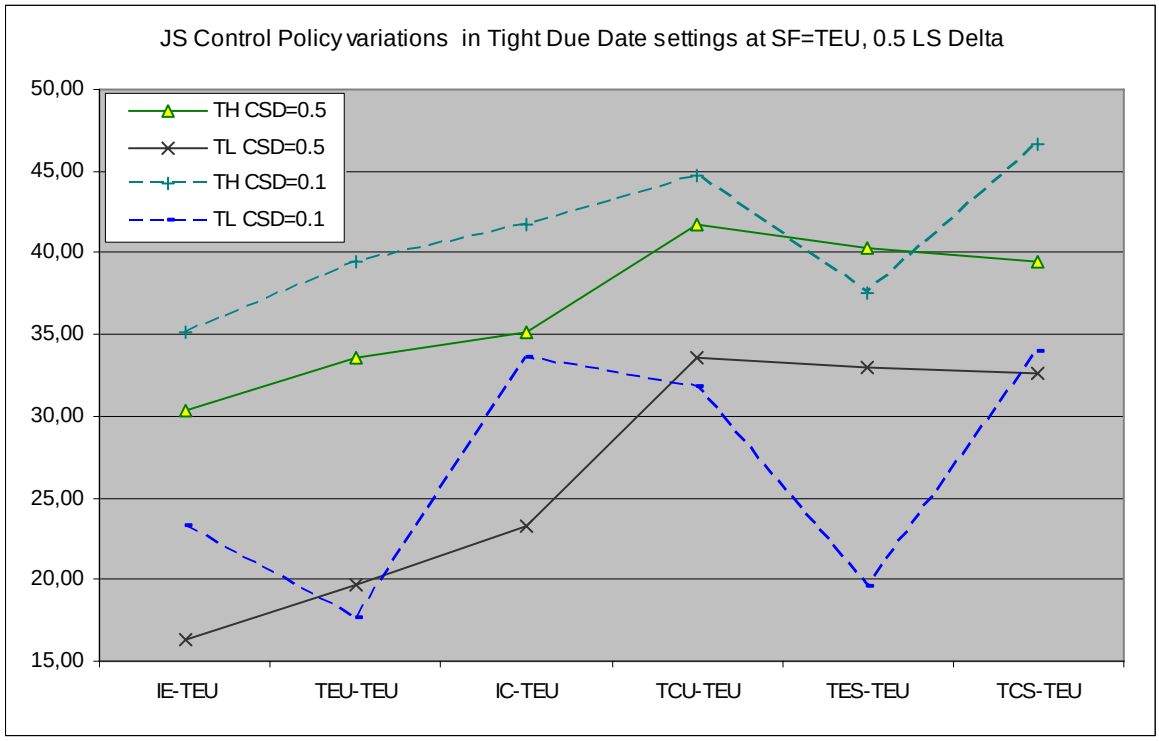

Figure 7.20. JS Control policy variations in Tight Due Date settings at SF=TEU, 0.5 LS Delta

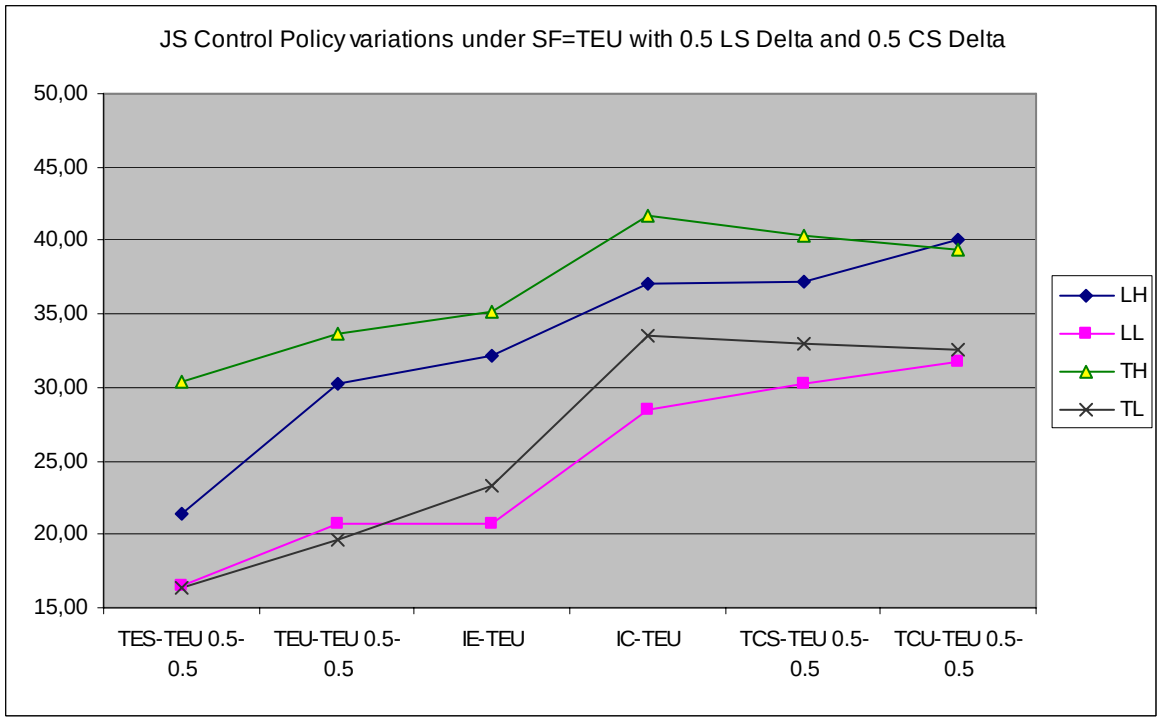

Figure 7.21. JS Control policy variations under SF=TEU with CS Delta=0.5 and LS Delta=0.5

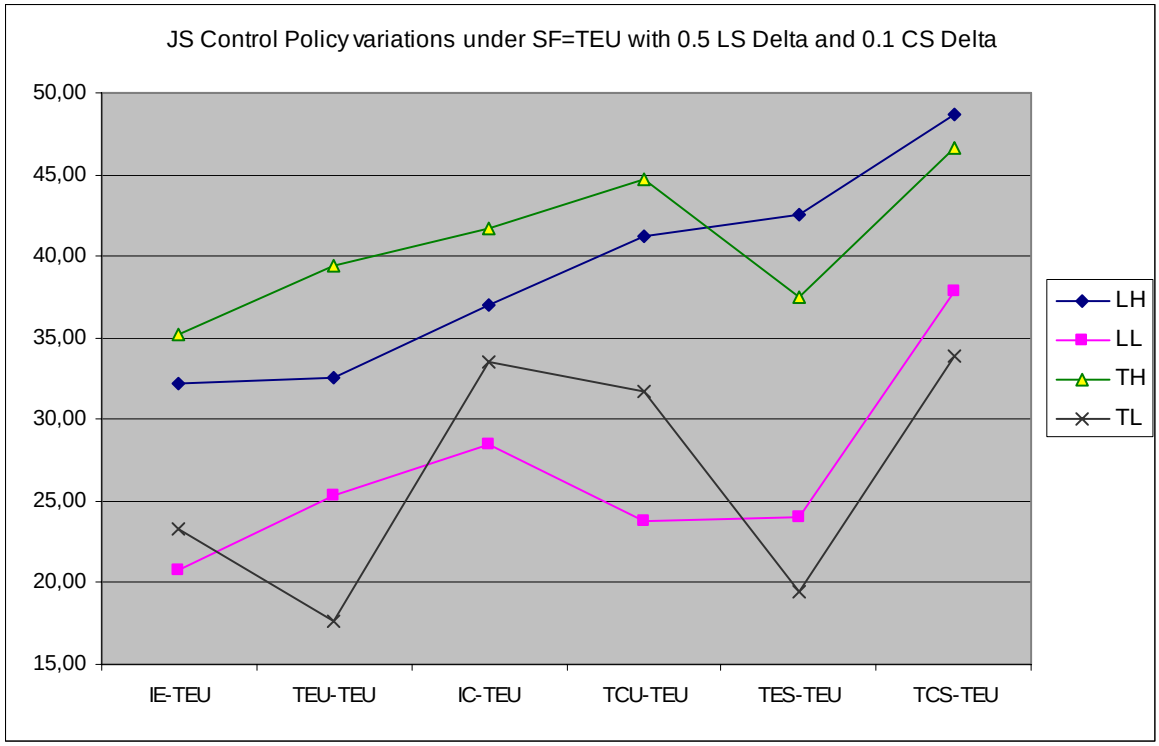

Figure 7.22. JS Control policy variations under SF=TEU with CS Delta=0.1 and LS Delta=0.5

The effect of decision time for the TES-TEU policy can be seen in Figure 7.23. In this figure the major gridlines on to the x-axis create 4 zones with LS Delta values 0.1, 0.25, 0.5, and 0.75 respectively. The CS delta and LS Delta values are appended together as the x-axis labels, and as an example 0.1-0.5 control policy indicates that there is an event based takes time control policy with 0.1 CS Delta implemented with a stable JS agent, and an event based takes time control policy with 0.5 LS Delta implemented with an unstable SF agent. Compared to TES-TES scenario the curves corresponding the tight due date settings, i.e. the TH and TL curves, are significantly separated from each other in this TES-TEU scenario. For 0.1 and 0.25 LS delta values an increase in CS delta also leads to poorer performance results for all settings. But from the zone 0.5 LS Delta on, an increase in CS Delta leads to better results for all problem settings except the LL setting at 0.75 CS and 0.75 LS Delta.

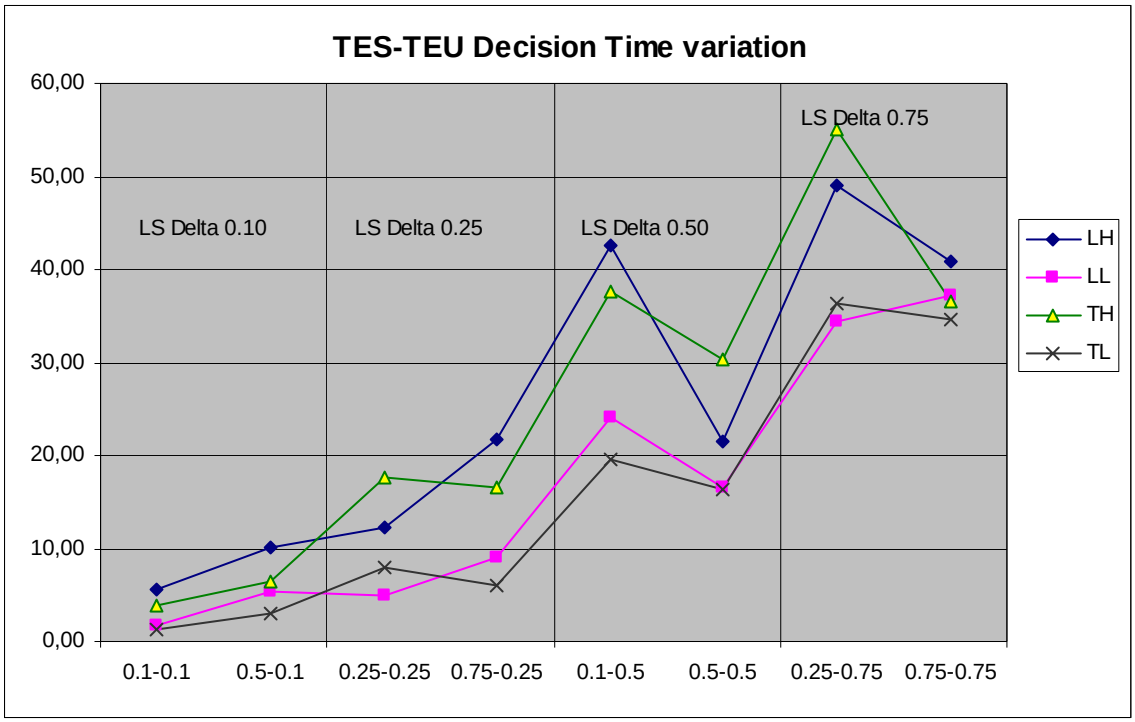

Figure 7.23. Decision Time variations under TES-TEU control policy

In Figure 7.24 one can see the effect of decision time under TEU-TEU control policy. As in the case of TES-TEU policy, for 0.1 and 0.25 LS delta values an increase in CS delta also leads to poorer performance results for all settings. But from the zone 0.5 LS Delta on, an increase in CS Delta again leads to better results for all problem settings except the LL setting at 0.75 CS and 0.75 LS Delta.

In Figure 7.25, and Figure 7.26 effect of decision time variation on TCS-TEU and TCU-TEU control policies can be observed.

They exhibit similar patterns. An increase in decision time in the shop floor leads to poor performance results. Again in the 0.5 LS Delta Zone, an increase in CS Delta leads to better results in the TCS job system policy for all problem setting environments. The same is true for loose due date and high utilization, i.e. LH, environment in the TCU job system policy. In all other settings an increase in CS or LS delta leads to poorer performance results.

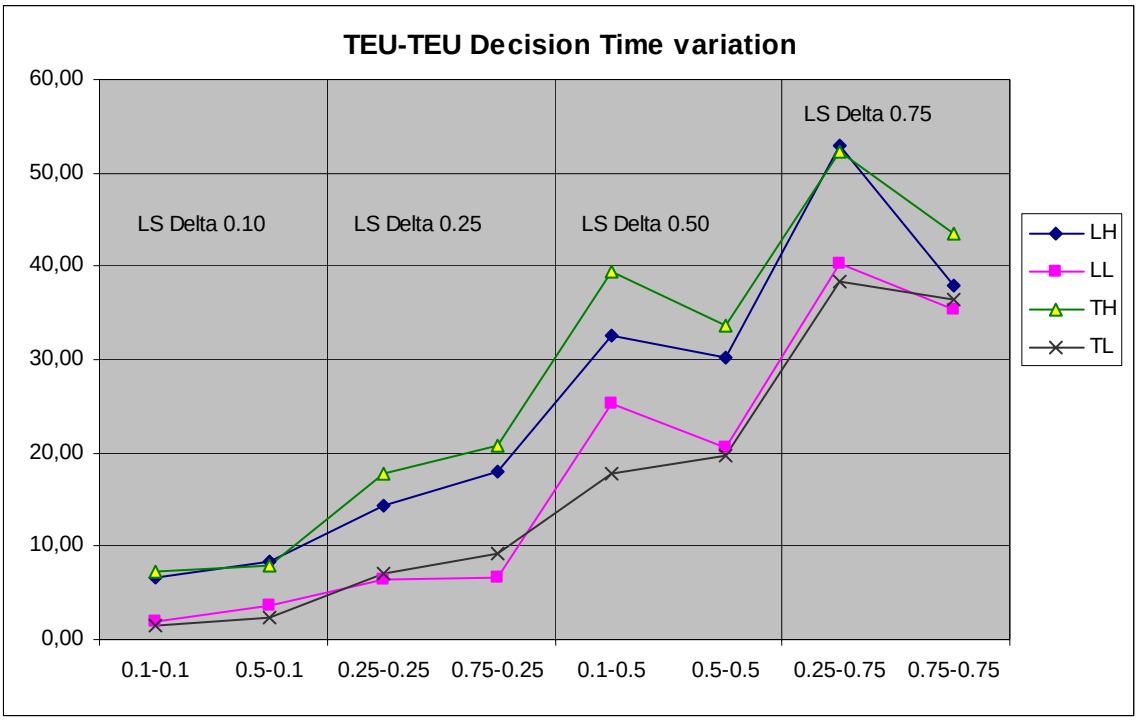

Figure 7.24. Decision Time variations under TEU-TEU control policy

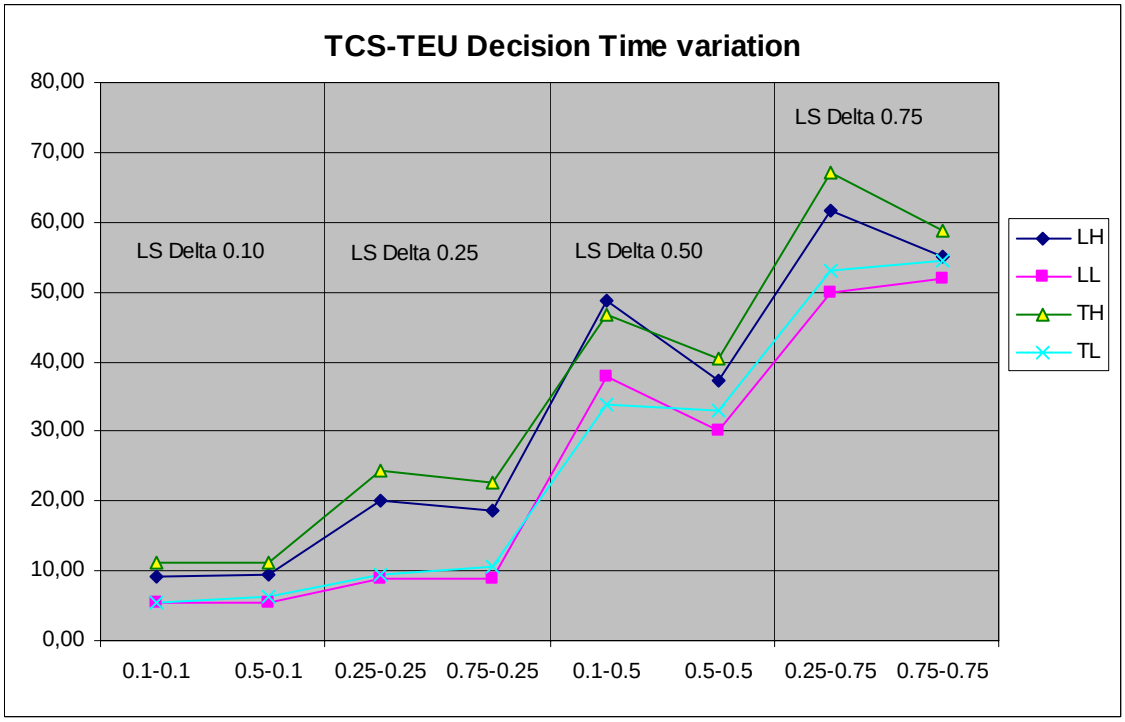

Figure 7.25. Decision Time variations under TCS-TEU control policy

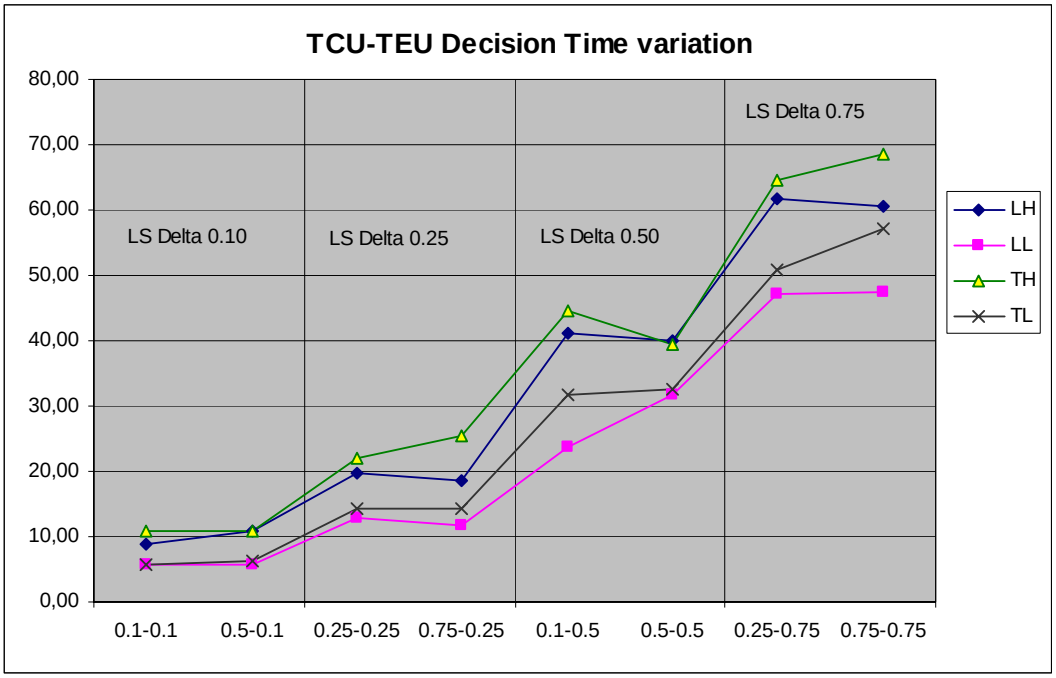

Figure 7.26. Decision Time variations under TCU-TEU control policy

The IE-TEU policy can be seen in Figure 7.10. It can be clearly seen that the increase in decision time leads to higher deviations from the utopic solutions, and therefore poorer results. The same pattern can be analyzed also for the IC-TEU policy graph shown in Figure 7.27.

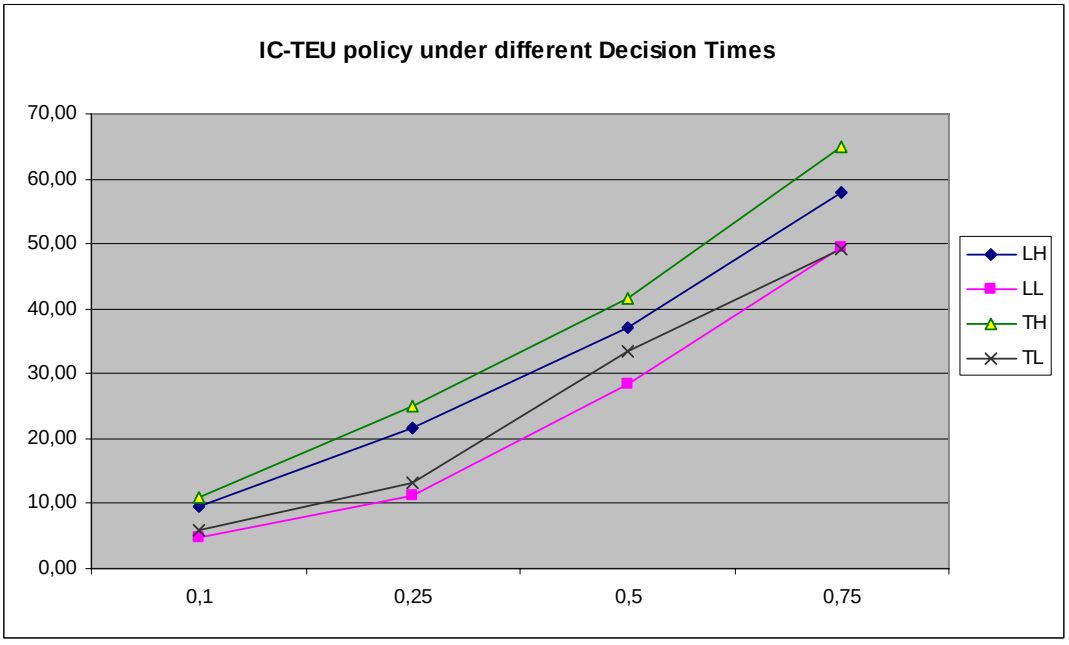

Figure 7.27. Decision Time variations under IC-TEU control policy

.

7.5. Effect of the Rescheduling Interval under IE based SF Policy

In this section the effect of the rescheduling interval will be analyzed. In Figure 7.28 and Figure 7.29 the IC-IE, and TCS-IE policy results are plotted against different CS Rescheduling Intervals. It can be clearly seen that an increase in the rescheduling interval clearly leads to higher deviations from the utopic values. They exhibit almost the same pattern. Although the TCU-IE graph is not shown in this section, it also exhibits very similar behavior like the TCS-IE policy.

In Figure 7.6 and Figure 7.7 in section 7.1 the variation of the decision time under different CS rescheduling intervals are shown for the TCS-IE and TCU-IE control policies. For both control policy the effect of the decision time is not significant with respect to the effect of the CS rescheduling interval. For every problem setting environment, one can see almost horizontal lines at different levels. The effect of the CS rescheduling interval, in these figures the values for this parameter are 5 and 20, plays the most important role in the differences. So for cyclic policies the change in the value of CS Rescheduling interval leads to higher deviations in results.

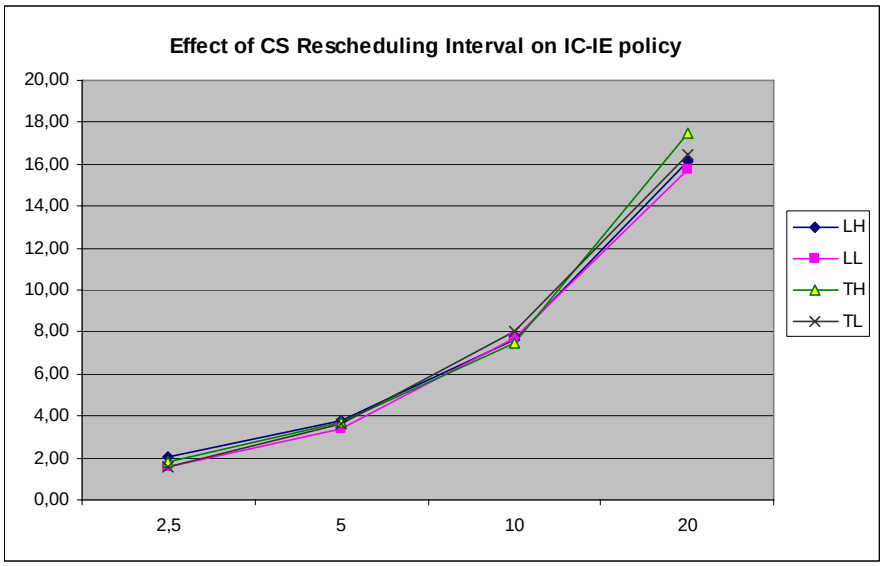

Figure 7.28. Effect of the CS Rescheduling Interval on IC-IE control policy

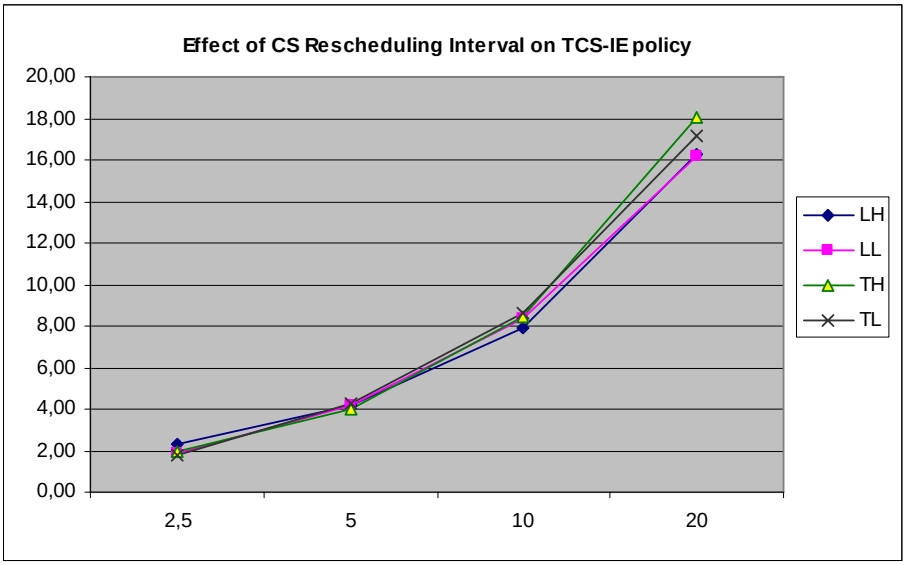

Figure 7.29. Effect of the CS Rescheduling Interval on TCS-IE control policy

7.6. Effect of the Rescheduling Interval under TES based SF Policy

In Figure 7.30, one can see the results for instantaneous cyclic job system control policy and event based takes time and stable agent control policy plotted against different CS Rescheduling values. For control policies which take time, the 0.5 multiple of the mean job interarrival time is used where applicable. The curves show a similar pattern for tight due date environments and loose due date and low utilization environments (LL) and an increase in CS Rescheduling value leads to higher deviations from the utopic solutions. For loose due date and high utilization environments increasing the rescheduling interval from 2.5 to 5 leads to better or similar results, but increasing the value further from 5 to 10 or 20, the deviations get higher for the IC-TES control policy. Almost the same behavior is present for LH environments in TCS-TES policy as shown in Figure 7.31. In all other settings an increase in CS rescheduling interval leads to higher deviations as in the case of IC-TES policy.

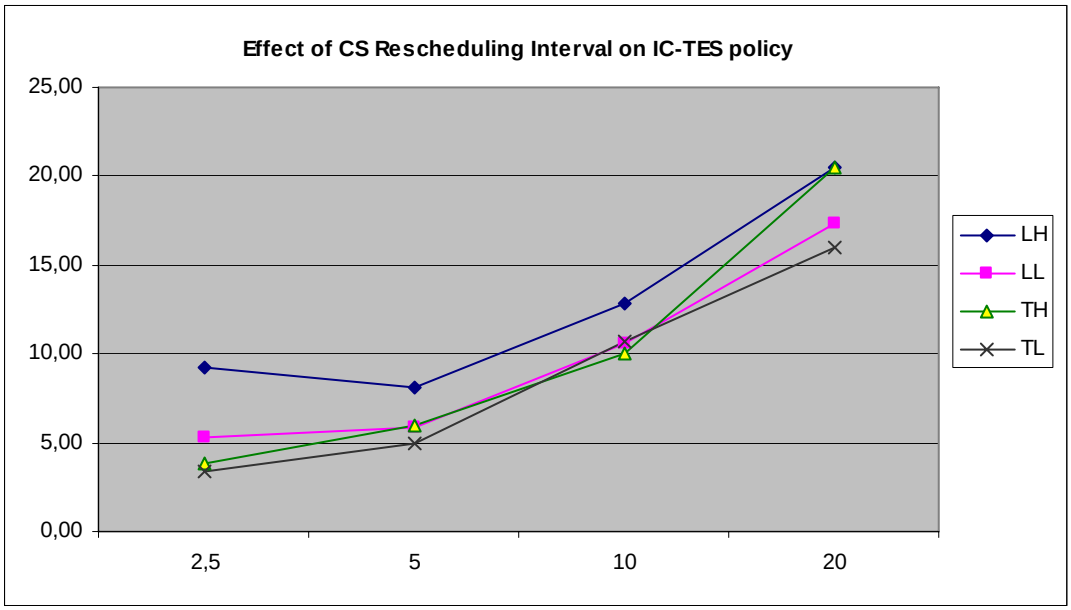

Figure 7.30. Effect of the CS Rescheduling Interval on IC-TES Policy

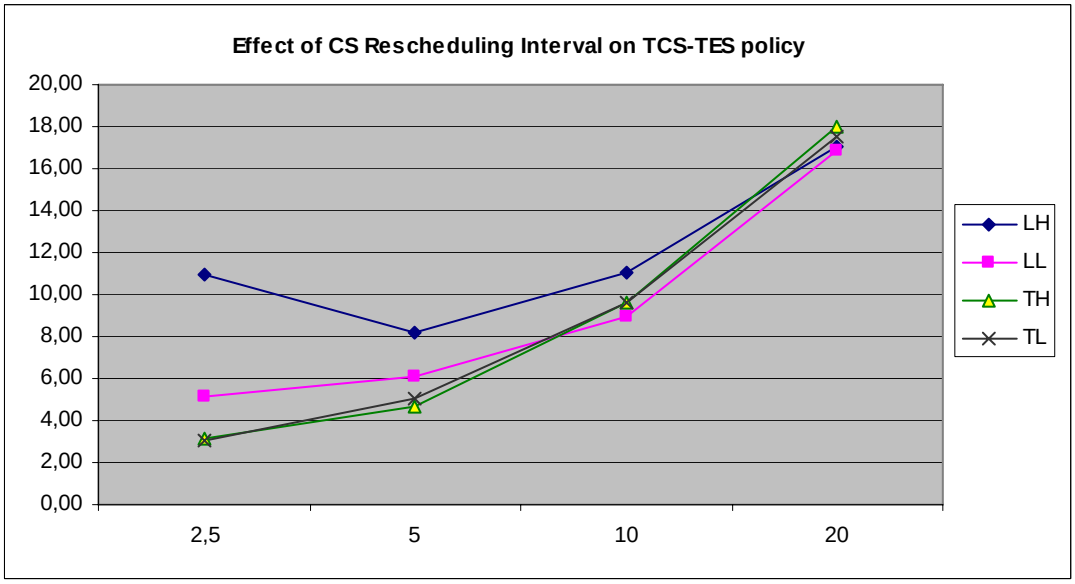

Figure 7.31. Effect of the CS Rescheduling Interval on TCS-TES Policy

The effect of the CS Rescheduling interval for TCU-TES policy is shown in Figure 7.32. In this control policy increase in rescheduling interval leads to poorer performance results for all problem environments. Another thing to note is that for LH environments the same amount of increase in rescheduling interval leads to smaller amount of deviations compared to the other problem setting environments, i.e. for LL, TH and TL.

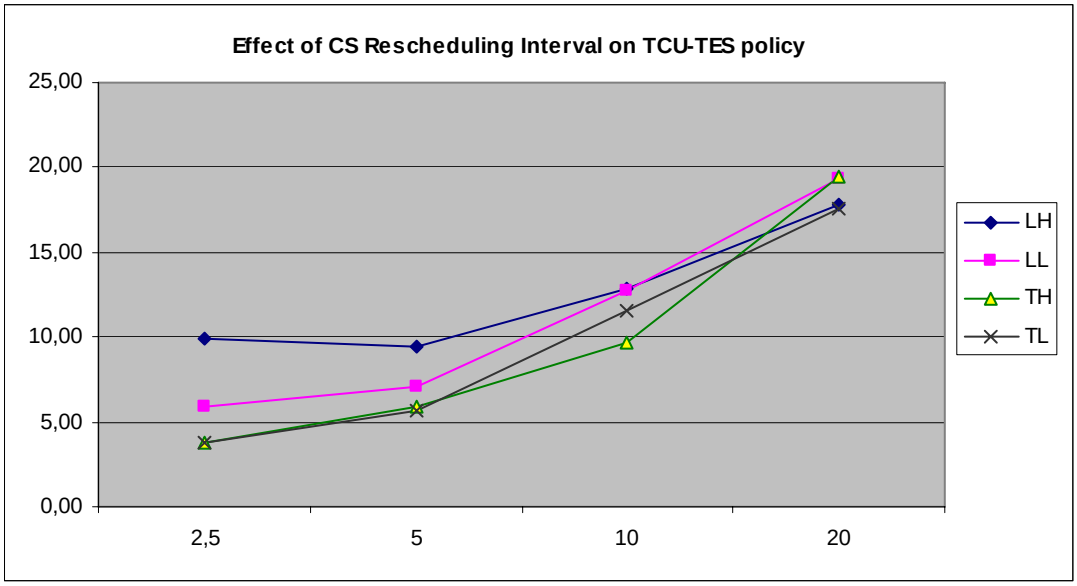

Figure 7.32. Effect of the CS Rescheduling Interval on TCU-TES Policy

7.7. Effect of the Rescheduling Interval under TEU based SF Policy

In Figure 7.33, one can see the results for instantaneous cyclic job system control policy and event based takes time and unstable agent control policy plotted against different CS Rescheduling values. For control policies which take time, the 0.5 multiple of the mean job interarrival time is used where applicable. All of the problem settings almost show similar pattern, and are increasing with an increasing rescheduling interval. For the TCS-TEU control policy shown in Figure 7.34, the high utilization and low utilization curves, i.e. the TH and LH curves and LL and TL curves form two couples, and they exhibit almost the same pattern. Again the increase in rescheduling interval leads to higher deviations from utopic solutions. The curves show a similar pattern for tight due date environments and loose due date and low utilization environment (LL) and an increase in CS Rescheduling value leads to higher deviations from the utopic solutions.

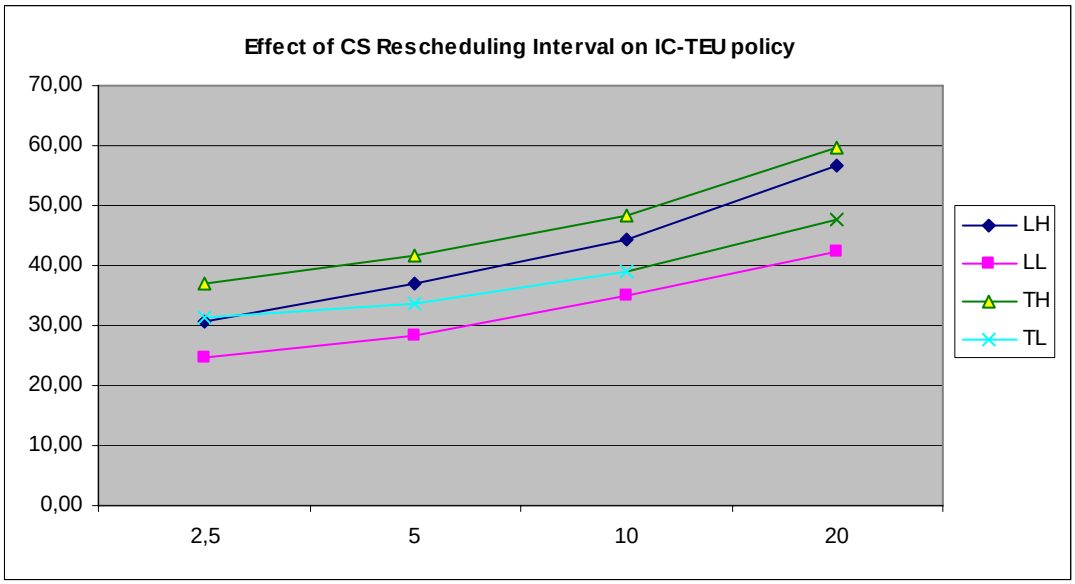

Figure 7.33. Effect of the CS Rescheduling Interval for IC-TEU policy

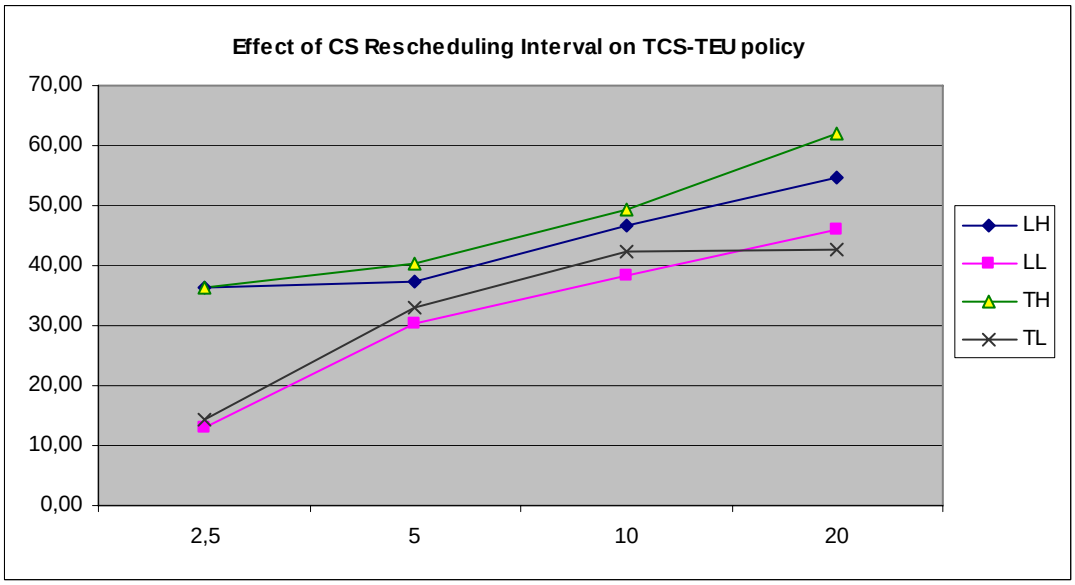

Figure 7.34. Effect of the CS Rescheduling Interval for TCS-TEU policy

For the TCU-TEU policy shown in Figure 7.35, the loose due date and high utilization environments show a significant increase in the results compared to the other LL, TH, and TL environments when the rescheduling interval increases from 2.5 to 5. But increasing the value further from 5 to 10 or 20, the high utilization curves, LH and TH, form a couple and show a similar pattern and the deviations increase. Also low utilization

curves LL and TL likewise form a couple and show an increasing trend with increasing CS Rescheduling interval.

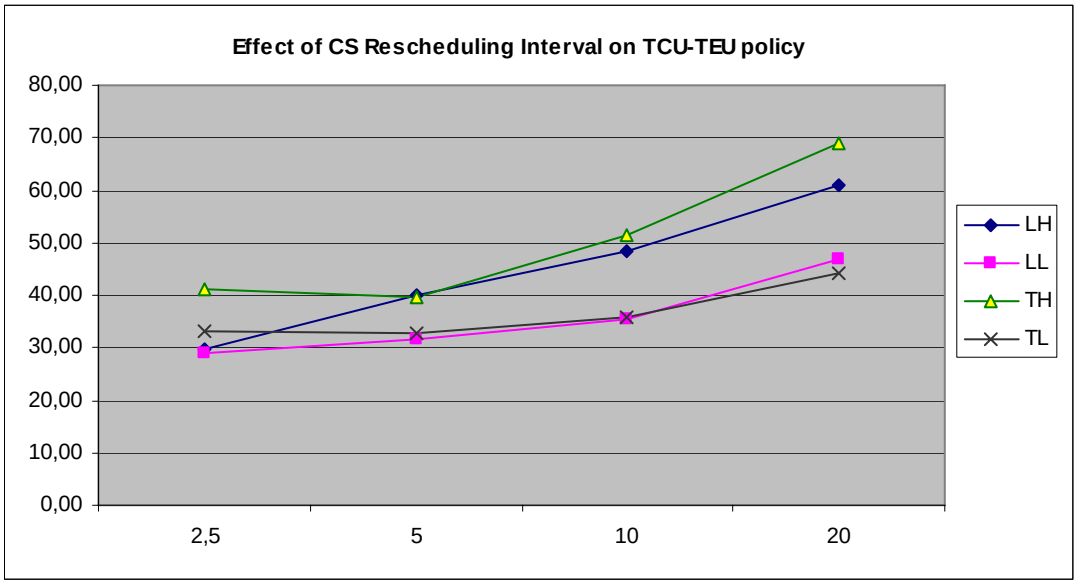

Figure 7.35. Effect of the CS Rescheduling Interval for TCU-TEU policy

7.8. Solution Procedure vs. Control Policies

Up to this point all the combinations of the shop floor and job system control policies have been studied in detail according to the values of the decision times of the respective agents, or the rescheduling intervals employed by the job system agents in the cyclic response policies. It can be stated that agent stability (AS), the response policy (RP), and the decision process (DP) itself building this control policies are three significant factors. For the sake of completeness, and to let the reader to fully visualize the effects of the policies, the solution procedure employed in the shop floor has been altered to the worst possible alternative.

The aim of this section is to compare the effects of the above mentioned factors to an instantaneous and responsive control policy with a random baseline solution procedure. In order to create a random baseline solution procedure the previously used preemptive EDD heuristic has been changed. A random selection rule is applied for the assigned operations to the machine. In order to ensure the completion of jobs, the random selection

is performed with 50% from the operations that have been started before, and with 50% from the operations that have been assigned to the machine. So the LS agent will randomly start or stop the operations assigned to it, but it will definitely start with 50% an operation, which is previously stopped in order to start another operation, if any available. This algorithm is named as IE RND and will be presented at the end of this section.

The overall results of the system with this algorithm, IE RND, employed in the LS agent are compared to all other shop floor control policies with the optimum solver employed in the LS agent, under all job system control policies. All the figures are plotted in the order of the averages of the results of the selected policy under all problem setting environments.

First the instantaneous and event based job system control policy case is studied and can be seen in Figure 7.36. In this setup the IE-IE RND policy leads to 3rd best result in Loose Due Date environments, after the best IE-IE policy, and IE-TES policy with 0.25 LS Delta decision time. It ranks as the 4th tight due date environments, but better than all TEU based SF policies with 0.25 and 0.5 LS Delta values. The IE-IE RND policy performs better than IE-TEU policy with 0.25 LS Delta equipped with optimum dispatch policy with 99.89% significance. So here one can see that just being responsive with the worst possible algorithm will perform better than the SF algorithms with TEU policy when there is a decision time introduced. It also works better than the TES based policies under LH or LL environments when the LS Delta is higher than 0.5 mean job interarrival time.

In Figure 7.37 one can see the results of the SF control policies under an IC based job system control policy. For LH environments the IC-IE RND works as the second best control policy after IC-IE policy, and ranks as the 3rd in all other remaining problem setting environments. That is for LH environments being responsive with a random dispatch algorithm works better than the compared takes time policies no matter stable or not. For LL, TH, and TL environments only the IC-TES with 0.25 LS Delta can beat this policy. So the IE-RND policy works better than the TES policies with LS Delta greater than 0.25, and all the TEU policies. It is also interesting to note under cyclic job system policy the difference between the optimum dispatch algorithm, and the random dispatch algorithm is not so high.

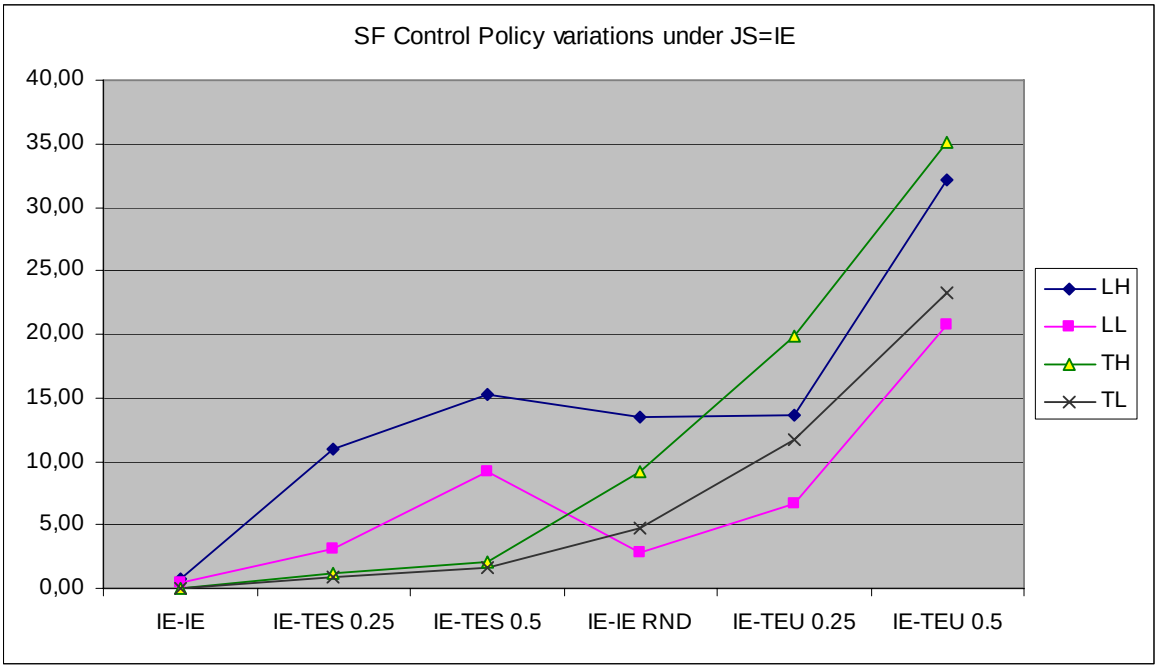

Figure 7.36. IE RND compared to other SF control policy variations under JS=IE

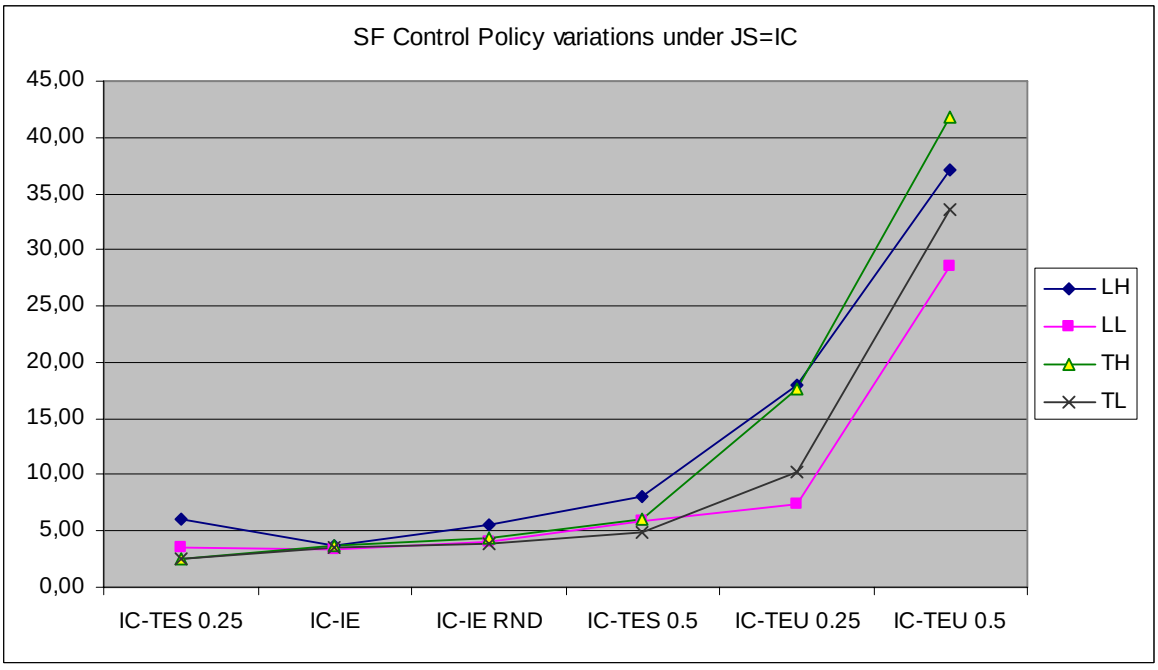

Figure 7.37. IE RND compared to other SF control policy variations under JS=IC

In Figure 7.38, the results of the SF control policies under an TES based job system control policy can be seen. Surprisingly the IE RND algorithm works as the 5th just before the TES-TEU 0.5 policy at the overall for all problem setting environments. In this case

since the job system policy is a TES based policy, the job system will send triggers twice compared to other SF policies in one job interarrival time, since it is event based. So the IE RND algorithm will work twice, which will make the result of this policy higher and higher. In this case for high utilization environments it works better than the TEU 0.5 SF policy. But it will generate the worst results for low utilization environments.

Also the TEU policy as the job system control policy is studied and shown in Figure 7.39. In this setting the IE RND policy performs as the 3rd best policy compared to the other SF policies after IE and TES 0.25 SF policies in the overall average for the problem setting environments. For Loose Due Date environments it works better than all the TEU based control policies with LS Delta values 0.25 and 0.5 and higher. It works better than all the TES based control policies with LS Delta values greater than 0.25. In the case of loose due date and low utilization LL environments it performs as the second best SF policy, and in general for low utilization environments under TEU based JS policy it leads to good results which are very close to the IE and TES 0.25 based SF policies.

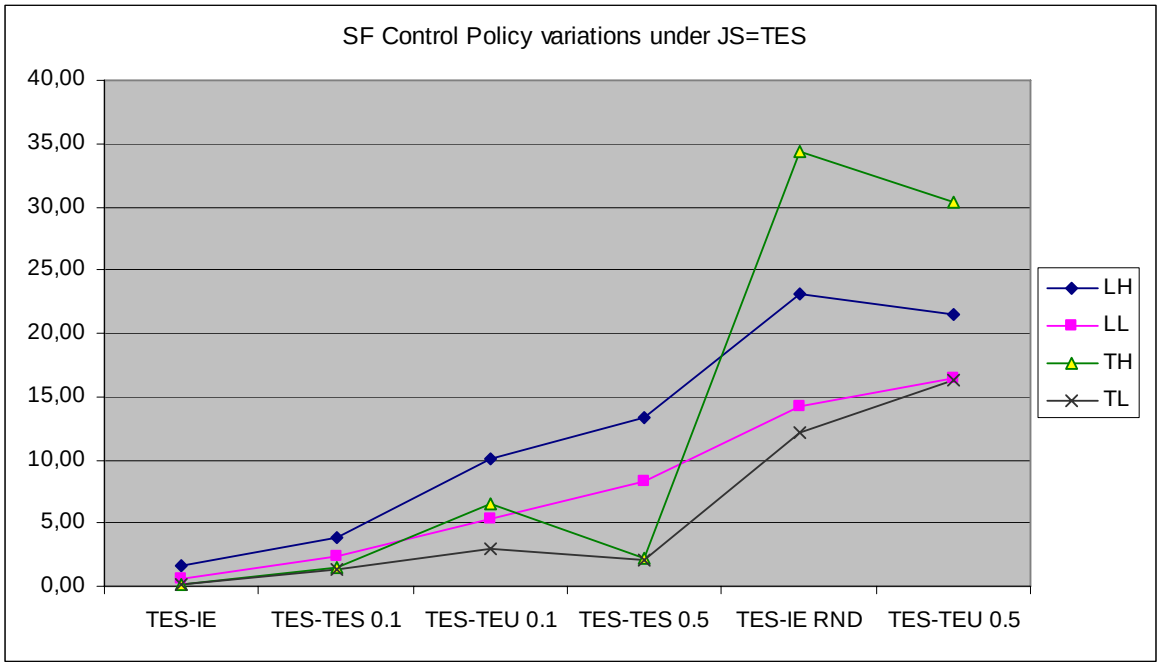

Figure 7.38. IE RND compared to other SF control policy variations under JS=TES

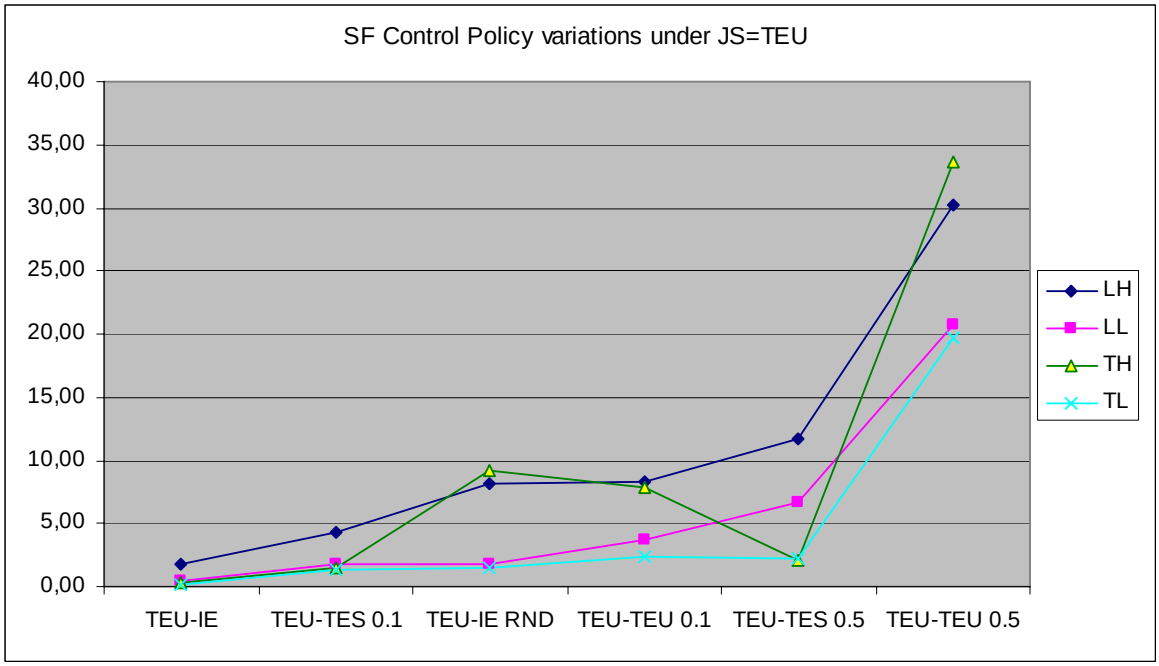

Figure 7.39. IE RND compared to other SF control policy variations under JS=TEU

The TCS based JS policy leads to similar results like the IC based JS policy and shown in Figure 7.40. Although the job system has a stable takes time control policy, and the random baseline algorithm will be called twice compared to IC, it does not lead to big deviations, since the interval is 5 times bigger than the case in TES based JS policy. The reason for this is that the CS rescheduling interval is taken as 5 multiples of the mean job interarrival time. In this setup the IE RND algorithm works as the 3rd best algorithm at the overall for all problem setting environments. The TCS-IE, TCS-TES 0.25 and TCS-IE RND policies almost lead to same results and almost create a horizontal line in the graph, so the results of the IE-RND algorithm is pretty good. So the IE-RND policy works better than the TES policies with LS Delta greater than 0.25, and all the TEU policies.

Finally the TCU based JS policy will be studied in Figure 7.41. In this setup it works as the second best policy after the TCU-IE policy at the overall for all problem setting environments. So introducing a random baseline algorithm as a dispatch algorithm at the LS agent, creates almost no effect under TCU based job system policy, and it performs better than all the takes time algorithms whether stable or unstable. The TCU-IE and TCU-IE RND policies almost lead to same results and almost create a horizontal line in the graph.

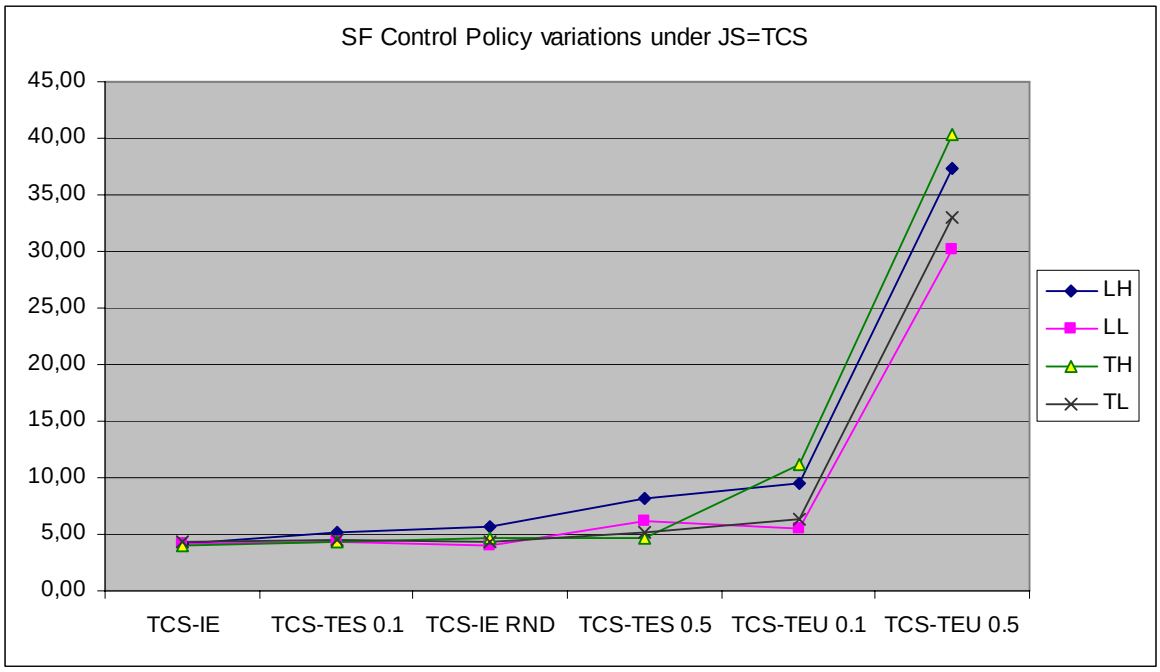

Figure 7.40. IE RND compared to other SF control policy variations under JS=TCS

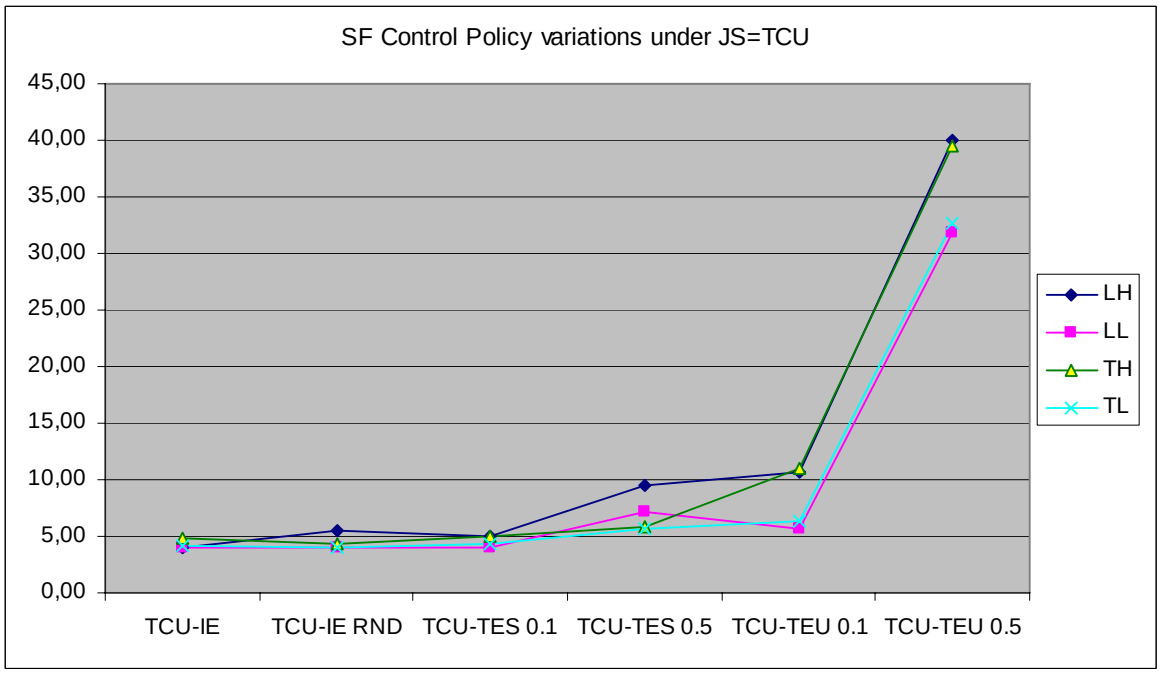

Figure 7.41. IE RND compared to other SF control policy variations under JS=TCU

7.8.1. The Random Baseline Solution Procedure : IE RND

To let the reader remind with the notation used in section 5, the following variables will be redefined in this section.

Let Qj^a : all operations of the jobs J^a , which are to be controlled by agent $a \in A$. Let $S_{qt}^a(\omega, c) = \{ F_q, C_q, Y_q, M_q, X_q \}$ denote the state of operation q at time t observed by agent a according to realization ω and control policy c, where

- F_q : start time of operation q
- C_q : completion time of operation q
- Y_q : percent completion of operation q , $Y_q \in [0,1]$
- M_q : assigned machine of operation q
- X_q : status of being processed or not of operation q, $X_q \in \{0, 1\}$

The algorithm selects the next random operation *qnext* with the following steps;

1. Set *qnext* = {}
2. Select an operation *q1* randomly from Qj^{LS} where $S_{q1t}^{LS}(\omega, c) = \{ F_{q1}, C_{q1}, Y_{q1}, M_{q1}, X_{q1} \}$ and $Y_{q1} = 0$
3. Select an operation *q2* randomly from Qj^{LS} where $S_{q2t}^{LS}(\omega, c) = \{ F_{q2}, C_{q2}, Y_{q2}, M_{q2}, X_{q2} \}$ and $F_{q2} >$ TIMEZERO and $0 < Y_{q2} < 1$
4. Select an operation *qnext* randomly from {*q1*,*q2*}

7.9. Effects of Control Policies at a Glance

In this section of this study, the effects of the control policy parameters are summarized with the following three dimensional graphic. In this figure x and y axis represent the control policies applied at both the CS and LS agents. The z axis represents the problem environments. Again the values for CS and LS delta the 0.5 multiple of job interarrival time is used, and the CS rescheduling interval is taken as five multiples of job interarrival time. The size of the spheres represents the performance value of the used

control policy, i.e. the difference between the simulation result and the utopic solution. The difference values are multiplied with a scaling factor in order to see the differences clearly.

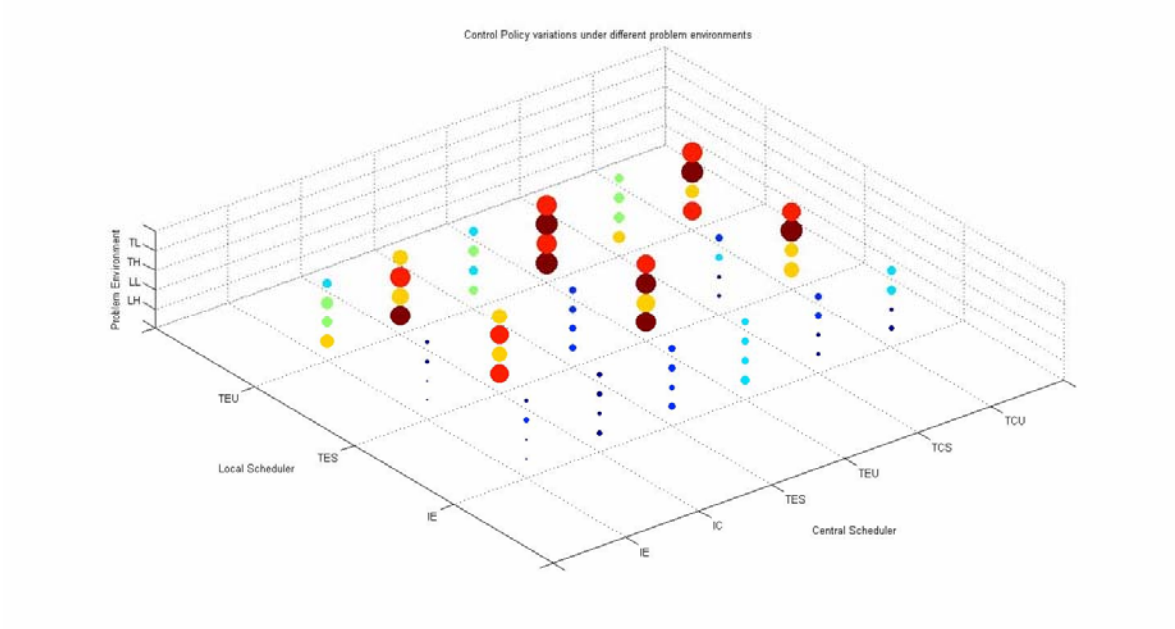

Figure 7.42. Control policy variations at a glance

In this figure we see that the instantaneous LS control policies perform almost the best in general. The same is also valid for the CS control policies. The takes time control policies perform poorer than instantaneous policies in general. It can also be seen that unstable control policies perform poorer than stable control policies in the LS agent regardless of the CS control policies. As a summary all the control policy effects of both agents at different problem environments can easily be seen in this figure.

8. CONCLUSION

In this study we have introduced a mathematical foundation and a framework for the modeling and analysis of agent based distributed scheduling systems. We defined the basic structure of an object oriented agent based system, and the issues in distributed systems. In order to explore the agent based structures deeply, a state based modeling approach is used.

This proposed mainframe is used in the development of the distributed agent based framework,, which is being used for testing. The concept is validated with this framework. From this implementation it can be seen that orthogonal aspects and interaction of agents with their object domains can efficiently be modeled with this framework.

We have been able to show that the set of control variables combined under the name of control policy have a significant impact on the overall system performance. The control policy parameters are build from the decision process, i.e. whether it is instantaneous or takes time, the Response Policy, i.e. whether it is cyclic or event based, and finally the agent stability, whether the respective agent will be stable or unstable during the decision process. The control policies implemented in the LS and CS agents have been tested in different problem setting environments which are classified according to their due date tightness and system utilization parameters.

In this study we showed that if the control policies of the LS and CS agents are not set up correctly for the system, even an optimum solution procedure employed in the LS agent may not guarantee a good solution. We also showed that the performance of these control policies change from one problem setting environment to another. It has been shown that introducing a decision time especially in the LS agents control policy leads in general to poor performance of the system.

We see that the effect of the CS rescheduling interval has an important impact to the system in cyclic response policies, and generally increasing the CS rescheduling interval leads to poorer performance of the system except for the loose due date high utilization (LH) environments under the cyclic response policy job system setups when the SF policy

is event based takes time policy with a stable agent structure, i.e. TES. On the other hand the impact of the rescheduling interval to the system is higher compared to the effect of the decision time.

Under takes time shop floor control policy, it has been observed that the cyclic job system response policies perform significantly better than the event based ones under loose due date environments. The contrary is valid as would be expected for all cases, i.e. for tight due date environments, the event based job system response policies perform significantly better than the cyclic ones. The tight due date environments lead to slightly better results than the loose due date environments, but this difference is not statistically significant. The low utilization environments lead to significantly better results than high utilization environments.

Especially in control policies which have both takes time policy for the LS and CS agents, we observe an interesting behaviour of the system. Generally when the LS Delta value is set as 0.5 multiple of mean job interarrival time we see that delaying the job submission in the CS agent to the LS agent may lead better performance for the overall system.

For the sake of completeness, and to let the reader fully visualize the effects of the control policies, the solution procedure employed in the shop floor has been altered to the worst possible alternative, i.e. to a random baseline solution procedure. It has been shown that introducing the decision time to the decision process leads to significantly poorer results compared to instantaneous policies, and even a responsive random dispatch policy may perform well better compared to a takes time control policy equipped with an optimum dispatch algorithm.

From these results we can say that when analyzing the scheduling problems, the employed control policy in the system is as much as important as the solution procedure itself, and an optimum solution procedure may not guarantee good results, when the control policy is not set up correctly. Therefore it would be more appropriate to analyze the problems with the system control policy setup as a whole and according to their classification, i.e. according to their due date tightness, or utilizations. With the random

baseline solution procedure we showed that under some conditions even randomly dispatched operations may lead to better results.

We also observe that the synchronization between the decision makers, i.e. between the CS and LS agent, also plays a role in the system performance. This phenomenon can be especially observed when both the CS and LS agents have takes time decision processes. From these findings we may also state that somehow there should be a harmony between the decision makers. As the future work, the synchronization of the decision time and the control policies with respect to the decision makers should be analyzed.

To summarize, we introduced the decision time effect to the literature, and setup a mathematical foundation and build an agent based framework in order to analyze its effects. We introduced three major control variables, and showed that they have a significant impact to the system performance as a whole, and even the optimum solution procedures may lead to poor performance when the control policy is not setup correctly. Finally we see a strong evidence that the harmony between the decision makers affects the system performance.

APPENDIX A. ICRON IMPLEMENTATION

A.1. ICRON as the Preferred Framework

There should be a generic framework to evaluate and implement the distributed scheduling concepts on it. This framework definitely should have a simulation environment, and especially a distributed simulation environment. There were two candidates for being selected as the framework for this study. The first one was the simulation and control framework developed by Kurşun 2001, and the second one was ICRON Planning and Optimization applications.

Kurşun 2001, constructed a distributed and object oriented simulation framework, and introduced a Virtual Manufacturing Component concept, which refers either to an object that represent the counterparts of real manufacturing components or to an object which has an interface to control the real manufacturing components. A system, which enables both controlling and simulating of any given manufacturing environment at any scale, from anywhere in the world via internet in an asynchronous manner, is developed. The proposed architecture has the ability to minimize the execution load of a complex simulation by distributing the simulation to many physical computers, when looked at the system simulation perspective. When looked from the control perspective, one will see that the proposed system enables the control of any manufacturing machine of a factory from any source, which can either be the web based clients, or a third-party enterprise-wide software running anywhere.

As the second alternative we had ICRON Planning and Optimization applications. ICRON is like a RAD (Rapid Application Development) tool, but a more proper name could be RSAD (Rapid Scheduling Application Development) tool. It has a very special module called GSAMS (Graphical Scheduling Algorithm Modeling System). With the help of this module one can easily develop any algorithm not limited to scheduling just by dragging and dropping basic or advanced nodes, and connecting them properly i.e. defining relations between them. By using GSAMS, users can generate and modify their scheduling algorithms without any programming language knowledge. GSAMS allows

users to drag and drop algorithmic units on to the design window, and visually construct scheduling algorithms. Using the services and functions of ICRON scheduler, the developed algorithm could be run, and the generated schedule could be viewed and analyzed.

In order to model an algorithm, the user works on nodes. Each node represents a particular operation on the current object. Nodes have input and output link points. Objects flow between the nodes through link points and all of the data manipulations are managed by node and link structures of the algorithm model.

Besides the basic nodes like Data source manipulation, Execution Flow, Input/Output, List manipulation, Messaging, Object model navigation, Reporting, System Construction and XML, it also has advanced nodes like Distributed processing, Mathematical Programming, Rule Engine, and Scheduling.

Another important point is the SFSim module in ICRON, it is a simulation module. There are SFResources, SimResources and SCHResources in ICRON. Actually SimResource refers to the manufacturing component itself, and SFResource could be a simulator or the interface to the manufacturing component. This concept is very similar to Kurşun's work. Besides the simulation framework ICRON has also a very strong scheduling framework. For these reasons ICRON is selected as the framework for this study.

In the study, Advanced Planning and Scheduling (APS) module of ICRON is used. In APS the system objects are classified into a set of distinct system components. Each component models a particular functionality of the scheduling system, and uses a well-defined communication mechanism to co-operative with other components. These system components are Physical System Component, Product System Component, Job System Component and Scheduling System Component.

The reason why we used ICRON as the development environment is that, the nodes in the system can be extended further to include other specialized functionality, such as state chart manipulations in our case.

A.2. State chart Implementation in Icron

In the proposed model, each object may have more than one state chart. A state chart defined for an object represents the possible named states of the object, events that this object may need to respond in the form of actions, and possible named state changes as a result of these actions. Each object state chart is the conceptual definition of the aspect in a sense, and introduced to the model as a visual algorithm. The state chart is a composition of state, event and action nodes.

- *State Node*: It has a name, i.e. state code and it has a conditional expression to check if an object is in this state or not. State nodes for an object in a state chart must be defined such that the conditional expression of only one state node returns true value at a time as stated in the property 1. It is a representation for the named states discussed in Chapter 3.

- *Event Node*: State nodes are followed by event nodes, and this forms a state event relation. This state event pair will be in RS as mentioned in Section 3.1. Event nodes represent the events in the system, and are just the name of the events. They connect the action nodes and state nodes. Event nodes act like conditional branching since for a particular state, different events may cause different actions, in addition to that, event evaluated in one state of the object may cause a different action than the same event in another state.

- *Action Node*: The decision on what action to take depends on the event and the current named state of the object. Action Nodes are defined to indicate the actions in the case of the event triggers a transition from current state. Action nodes are designed to execute an expression in its field. The action nodes follow the event nodes, and form an event action pair from the event action relations RE.

A state chart represents the current situation of the linked object at an instance. The condition expression in a state node is a function of an attribute or a combination of attributes of the current object. It actually checks a set of assignments to the states and

properties of the object, and evaluates its current named state accordingly. Object may be in one of the named states and available to perform an activity in case of a specific event received i.e. event is related to its current named state or may already perform an activity which may simultaneously or consequently change the current named state of the object. While the object is executing a particular operation in the action node it may be unstable state, thus unavailable to perform another process.

The state charts and all related definitions are represented in “State Chart Definitions” of an object in Icron. An example can be seen in Figure A.1.

In Figure A.1, one can identify the state chart definitions of a “Schedule” object. In Icron every algorithm has its own context, and the context of an algorithm can be seen as the string between the brackets in Figure A.1. The algorithms listed in this section are grouped according to their context. The algorithms can be either “SC”, “SC_Action”, or “SC_MSG” type of algorithms, which can be described as the followings;

- SC: It states that the context of the algorithm is a state chart. So when looking for the state charts of an object, one can easily find all from the list of algorithms with a context of “SC”
- SC_Action: It states that the context of the algorithm is an Action, i.e. it includes operations which may lead to state transitions.
- SC_MSG: It states that the context of the algorithm is a Message type, i.e. the execution of this algorithm will return a message in XML format.

As an example for the algorithms with a “SC” type context, a declarative model of a state chart of a machine object in the shop floor in Icron can be seen in Figure A.2. It is a state chart which is related with core concerns of a machine object. As seen from the figure the machine object may live in one of the “idle”, “inuse” and “broken” named states.

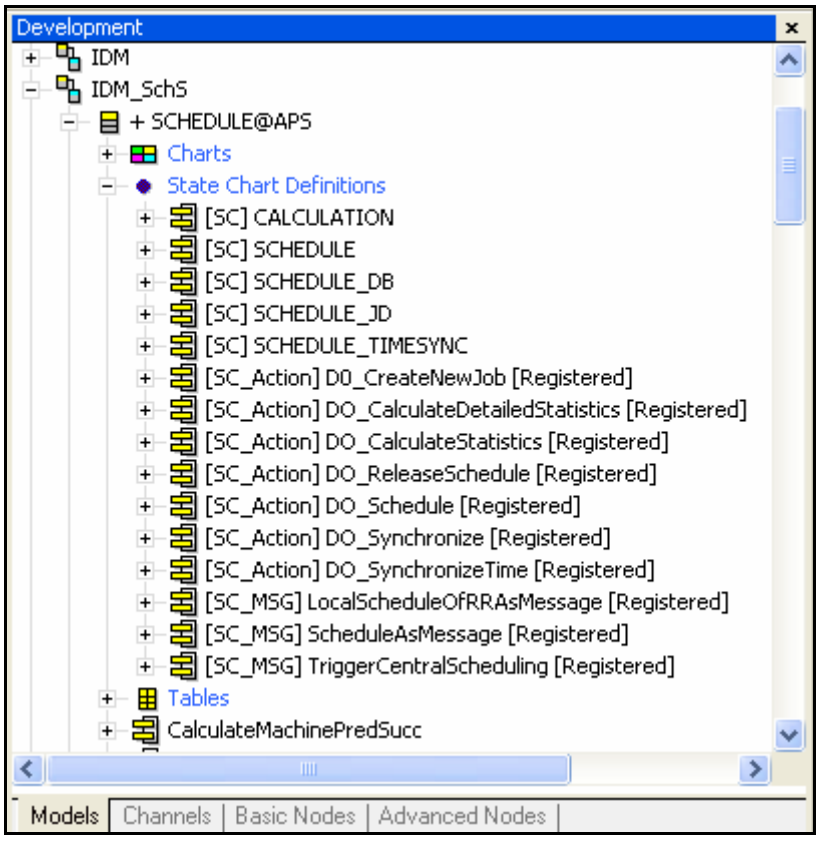

Figure A.1. A view of the development tab in Icron

The ”TheCurrentBreakdown” and “TheCurrentOperation” properties of the object are used to denote the current named state of the machine object. If the machine is not broken and it is not processing an operation, then one can say that the machine is in a “idle” state. If there is an operation which is currently being processed on the machine, i.e. the “TheCurrentOperation” attribute is set to 1, then the object is in “inuse” state. Finally if the “TheCurrentBreakdown” attribute is set to 1, then it means in the real life that the machine is broken, so its object will be in “broken” state. The lines connecting all these nodes in Figure A.2 represents the state event and event action relations, and the whole represent a decision map of what to do in response to an event in the system. So when the state chart is activated with an OperationStart event, and when the machine is in idle state, then in response to this event, the action “Do_OperationStart” will be executed. With the execution of the “Do_OperationStart” action, the machine object will make a state transition to the “InUSE” state. However when the machine is in broken state and the state chart is activated with the very same OperationStart event, then there will be no action executed in response to this event.

All possible responses to events are defined in the form of actions of the object in such a way that object performs one possible process at a time in each state chart, thus multiple processes which can be triggered simultaneously are excluded. For example, in Figure A.2 consider the machine is in inuse state. If the operation stop event occurs it will stop the current operation and make transition to idle state; and if the breakdown event occurs, it will make a transition to the broken state. These operation stop and breakdown events can not be processed at the same time. The event which occurs first is considered and other is not allowed to be performed. It would be appropriate also to study the state chart of an operation after studying the state chart of a machine.

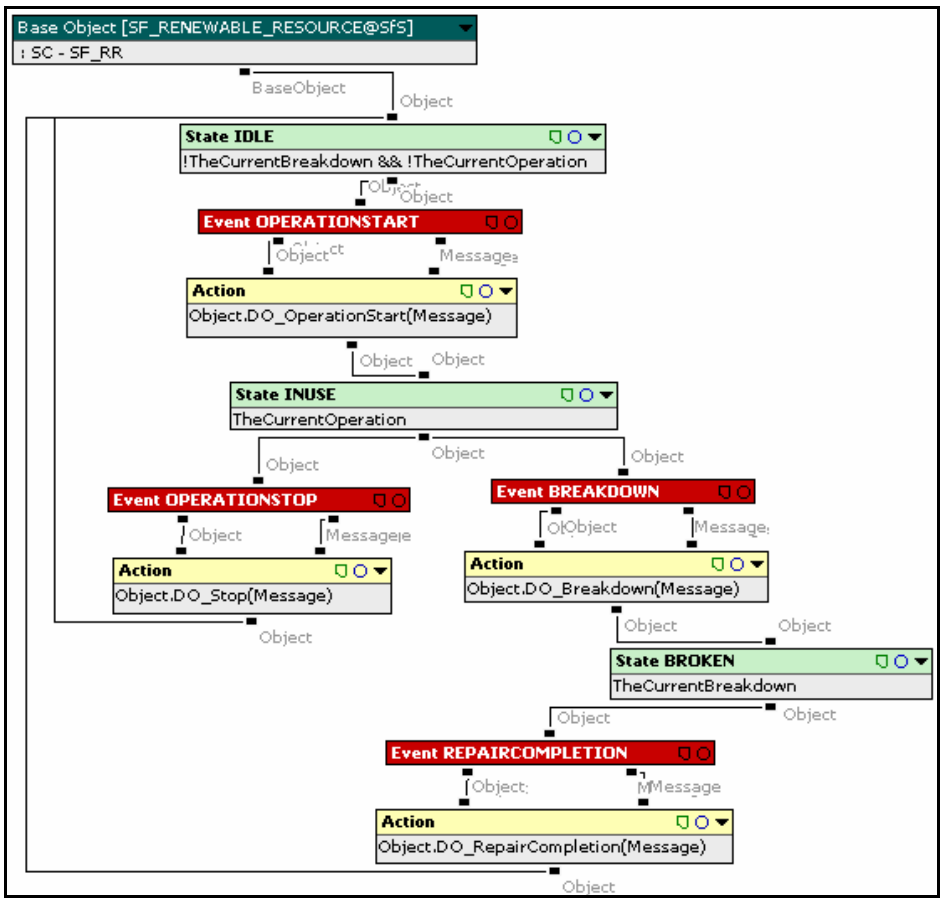

Figure A.2. The state chart of the machine object in the shop floor

In Figure A.3, the state chart of an operation in the shop floor can be seen. In this state chart one can see that the operation object can be in one of the “Available”, “Being_Processed”, “Closed”, and “Stopped” named states. The ”IsAvailable”, “Stopped”,

"Closed", and "RealizedST" properties of the object are used to denote the current named state of the operation object. If the "isAvailable" property is set to true, then the operation is in "available" state. If the "stopped" and "closed" properties are set to false, and the "RealizedST", i.e. the realized start time is greater than the default TIMEZERO value, then the operation is said to be in Being_Prcessed state. This means in the real life that the current operation is being processed. When the state chart is activated with an OperationStart event, and when the operation is in available state, then in response to this event, the action "DoStart" will be executed. Upon the execution of this action, the object makes a transition to the "Being_Processed" state.

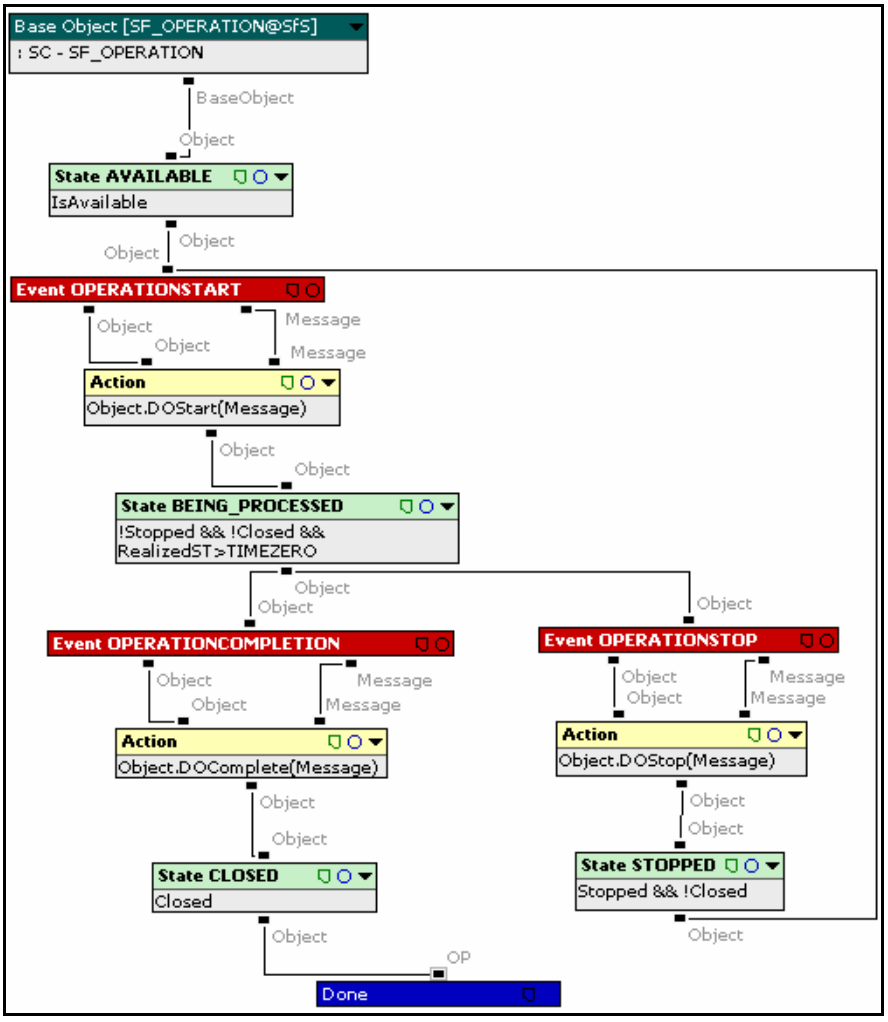

Figure A.3. The state chart of an operation object in the shop floor

A.2.1. Aspects of Objects as State charts

As mentioned previously, each object may have many aspects related to the scheduling, data and communication. The activities of an object are separated into possible distinct aspects and defined in respective state charts which can be run concurrently. With the help of the aspect oriented modeling approach one can easily express the crosscutting concerns with state charts and furthermore reuse these state charts in other objects' aspects. For example synchronization with the database of the current status of the objects is a very common process. When one considers a "Breakdown" event arriving to the SF_Renewable_resource_Agent, the corresponding HandleBreakdown MEP will first activate the state chart SF_RR, as shown in Figure A.2, with the Breakdown event, and consequently after the execution of the activate state chart process, the MEP will now activate the SF_RR_DB state chart, shown in Figure A.4, with the "UpdateDB" event in order to reflect the changes of the object into the database.

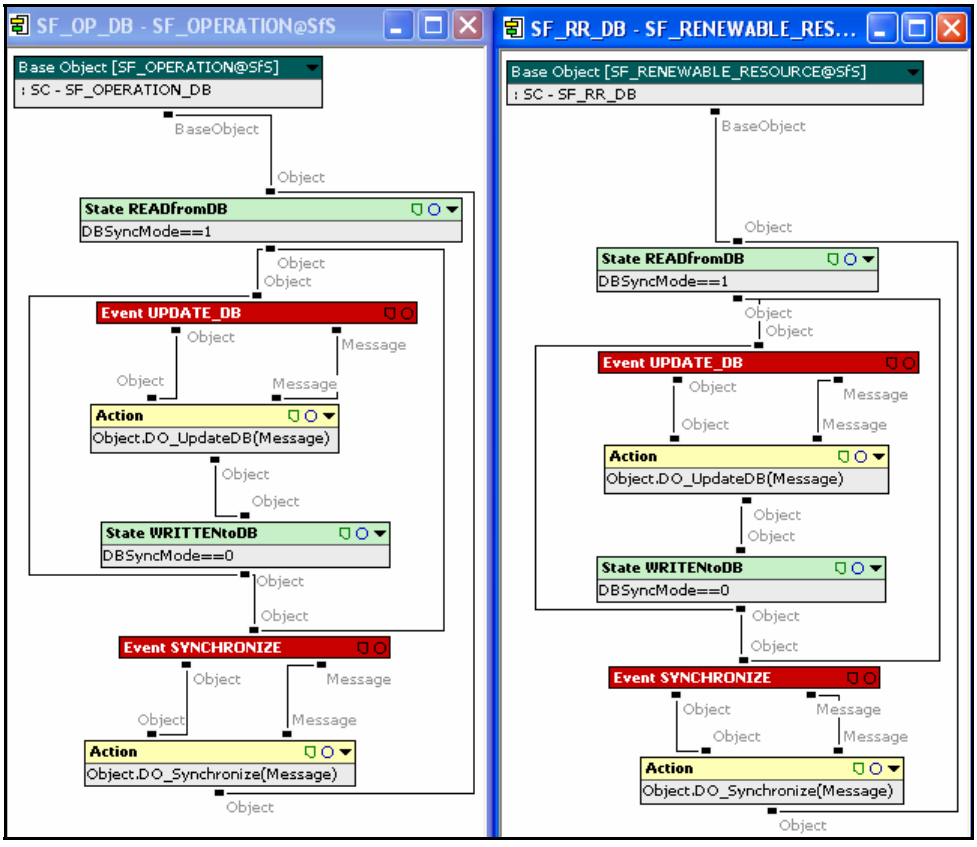

Figure A.4. SF_OPERATION_DB, and SF_RR_DB state charts of an operation and a renewable resource object in the shop floor

In Figure A.4, we see the database aspect of the operation object, and the renewable resource object in the shop floor. As one can easily see, the two state charts are almost the same, except the base object, which one can identify from the first node of both state charts. The state chart on the left identifies data store operations of an operation object, whereas the right one identifies data store operations of a renewable resource in the shop floor. Although the action nodes in both state charts are the same, as DO_UpdateDB, and DO_Synchronize, the workings of these actions are not identical. These actions are defined in different ways for each of the object, thanks to the polymorphism. The way to manipulate data on the database may change from object to object, or from time to time. So by just modifying the function identified in the action nodes the system will comply with the changes.

An additional scenario is that there exists other processes related to the data of the present object, i.e. the data of the object is shared in data store, and the instance of the mentioned data may be transformed by those processes, so the object needs to synchronize its current state by loading the data or part of the data from data store. This activity is triggered by the synchronize event.

There may be also some events which may not trigger a state transition but an application of some action in the current state of the object. In the state chart shown in Figure A.5 the object will perform an activity in response to a specific event and during the operation period (in action node), the object will not change its present state depending on the definition of the state, i.e. the condition is set to 1 which means it will be always true, since it does not evaluate anything against 1. In other words since the state of the object does not change during the execution of the action, the object remains stable, and so it is ready for another synchronize event to perform consequent synchronization action. In the case of the state chart shown in Figure A.5, the shop floor system object does not necessarily wait until the end of its action in order to accept a similar event to handle.

As it is previously mentioned the condition expression of the state of the Sfs_DB state chart is always true, however if it were the case that the respective attributes to define the state is processed during the activity then the object will be in an unstable state (not in the state defined in the state node). So for this example the condition expression may be

provided some kind of functions related to transformations in the action node. These kinds of separations are essential for mutual exclusion algorithms which are defined to prevent multiple processes on a shared data of an object.

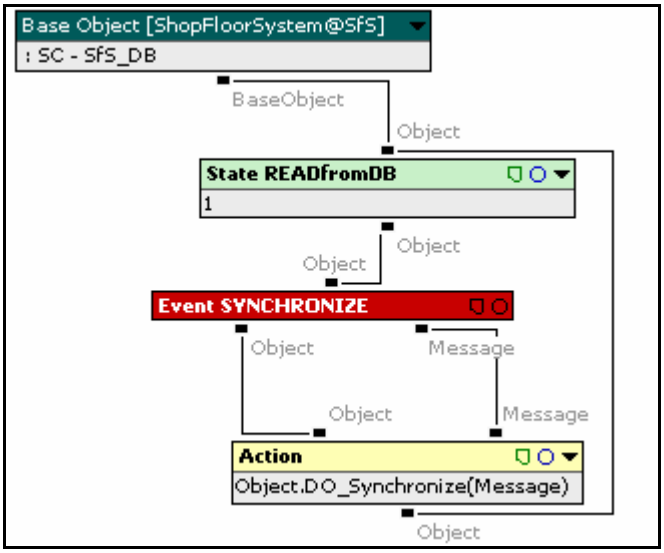

Figure A.5. Sfs_DB State chart of the shop floor system object.

In order to summarize the discussion about the aspects of the state charts it would be appropriate to give the state chart models of a renewable resource, i.e. a machine, from different objects point of view. A renewable resource object has a representation in the SFSim world, i.e. as a SIM renewable resource object in the SFSim domain, and its state chart can be seen in Figure A.6.

The SIM renewable resource object holds the real life data of the renewable resources in the shop floor system. In the SF world, the SF Renewable Resource object represents the same machine from the control point of view. In this case this object provides a control interface to the real life machine, and in the shop floor system these software components enable one to combine the real life data of the machines with the data of other scheduling entities of the shop floor environment. The state chart of the SF renewable resource object is previously discussed in Figure A.2. Finally from the scheduling point of view the same machine is represented by the SCH Renewable Resource object. The scheduling aspect of a renewable resource is represented in Figure A.7.

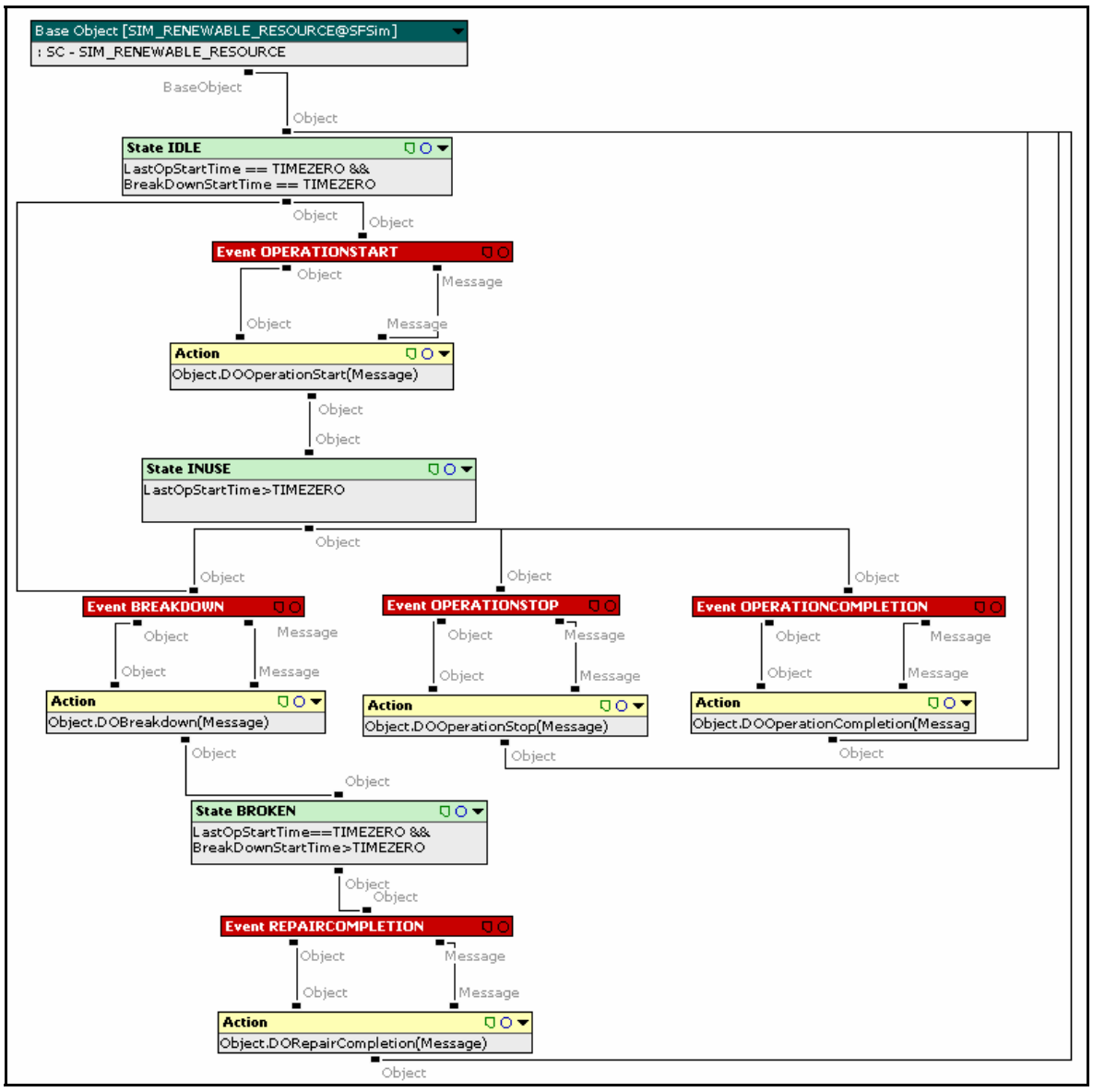

Figure A.6. The state chart of the SIM renewable resource in the SFSim domain

In Figure A.2, Figure A.6, and Figure A.7 different state charts of a machine which is represented with the help of different objects, and from different view points can be seen. The issues discussed in Chapter 4, are applicable in the representation of the machine in different objects.

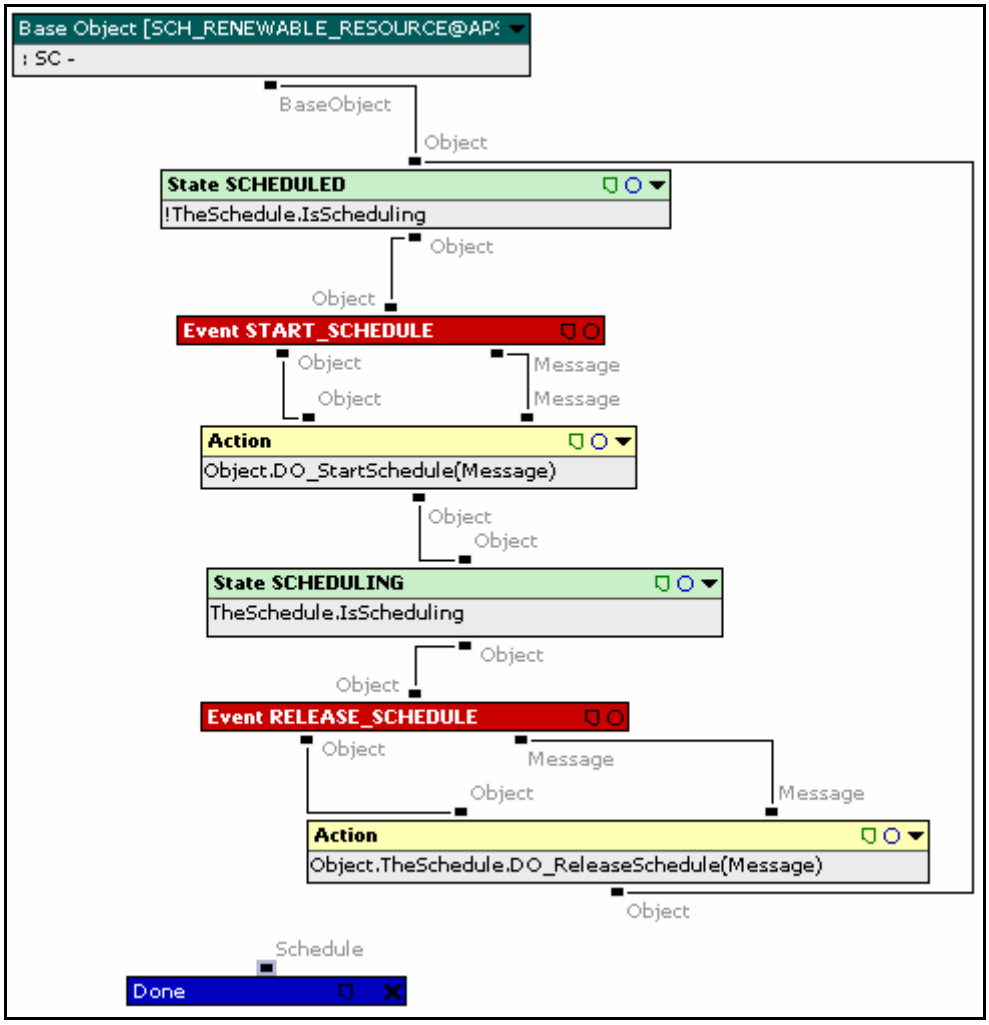

Figure A.7. State chart of the SCH Renewable Resource object

A.3. MEP Implementation in Icron

Communication between the agents is managed through messaging. In order to process a message from an agent the Message to Event Processors are used. Agents listen to their registered channels, and on the arrival of a message to their channel, they first identify the message, and if the context of the message is relevant to the listener agent then the message is processed. The agent may or may not respond to a message depending on the context of the message. If there is no MEP defined for this message, then it simply ignores the received message. MEPs of an agent are classified according to the context of the messages that agent processes, and each MEP is related with a specific message or a message set. MEP first decides which objects are related to the message. Then it maps the message to a respective "event" for the object, and triggers the state chart of the related

aspect of the object. MEP's are like functions, and have a sequence of actions to be accomplished. So by the execution of the MEP function, it may trigger or activate a state chart, and/or pass the message to another object or function with the help of the *Dispatch* function.

In this framework, an event is an interpretation of a received message in the context of a state chart, thus by using the proposed mechanism, messages are allowed to be reinterpreted in different aspects as corresponding events and dependencies between state charts are reduced. The "Activate State chart" and "Dispatch Message" nodes are specific nodes for the MEP's.

A.3.1. Activate State chart Node

Each MEP is related to an object or a group of object. In each MEP the event codes for the corresponding state chart of the related aspects are defined, and the message is processed for a specific state chart with a specific event by Activate State chart node. In Figure A.8 the Activate State Chart Node is shown. This node takes an object and a message as input, and by supplying the object to this node, in its user interface it finds all the state charts, and populates the "State Chart Combo Code" field, and all related events regarding to these state charts will be populated in the "Event Code" field. With the help of its graphical user interface, one can select the desired state chart, i.e. the combo field represents all the state charts regarding to the object, and upon selecting the state chart, all the events of the selected state chart are populated in the "Event Code" field.

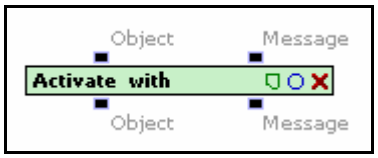

Figure A.8. Activate State Chart Node

The Activate State Chart Node executes the ActivateSC algorithm that is discussed in section 3.2.3.1. The working of this node will be discussed in the following section.

A.3.1.1. Working of the ActivateSC Function: The ActivateSC function activates the state chart *i* of object *o*, with first finding its current named state. Then according to the supplied event *e*, the system begins to execute the state chart as an algorithm starting from the current named state, and search for the appropriate state event relation to follow. Finally it evaluates the proper action according to the corresponding scheme according to the event action relation.

As an example if the ActivateSC is called for the state chart of a machine object as seen in Figure A.2, with the event breakdown. The function first finds the named state of the machine, say that it is in INUSE state. Then the relations from this named state to the next possible named states are examined, i.e. the states of idle, and BROKEN. It can be seen that the state chart shows that a relation between the INUSE state and BROKEN state exists, when there is an event called Breakdown. So in this relation, the sequence of execution steps shows an action to be executed, in this case, the DOBreakDown action, and it will be executed. After this transformation function is executed the object makes a state transition from its current state to its new named state, i.e. in this case to the BROKEN state.

All possible transformations of an object are defined as the action of the object. The actions of objects are defined in such a way that the object performs one possible process at a time, thus multiple processes which can be triggered simultaneously are excluded. In other words, the action of an object identifies the process which is mutually executed and the actions that can be concurrently performed are processed in distinct state charts.

As an example, consider again a machine object in in-use state. If the “operation stop” event occurs it will stop the current operation and make transition to idle state; and if the “breakdown” event occurs, it makes a transition to the broken state. These “operation stop” and “breakdown” events can not be processed at the same time. The event which occurs first is considered and other is not allowed to be performed.

In summary, the ActivateSC function may or may not lead to a state transition change. But if there will be state transformation, then the system will also verify itself in the sense of the named states. After a state transition, the object will be in its new named

state, but if the newly arrived state is not present in the current state chart, then this will indicate a problem in the state chart, i.e. in the model. So in this way a kind of verification is always performed by the execution of ActivateSC function.

A.3.2. Dispatch Message Node

The dispatch function was discussed in 3.2.2, the implementation of the dispatch function in Icron is realized with the help of the dispatch message node. In MEP's, the agent which received a message, decides on certain actions in response to this message with the help of state charts, and in order to deploy the decisions undertaken, it needs a dispatching mechanism. In order to send a message to a receiver, i.e. to an agent, the dispatch node is used. This node takes an agent as an input, and by supplying the agent to this node, in its user interface it finds all the dispatch message codes, and populates the "Message Code" combo field. In addition to that, the node needs the condition, message content, and synchronization mode parameters.

- Message Code: The message code can be selected from a combo field, and it indicates which specific message will be dispatched, i.e. which dispatch algorithm will be executed.
- Condition: This field specifies in which condition the message should be dispatched, generally it is set as true, i.e. every time the message will be dispatched.
- Message Content: In this field, one can either supply the content as a string, or can supply an algorithm with a context of SC_MSG, which in turn of its execution will return a message content.

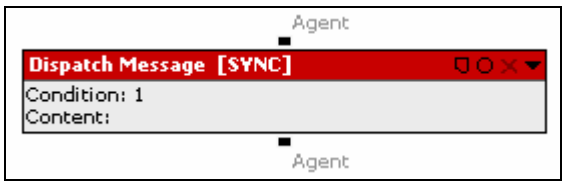

Figure A.9. Dispatch message node

During the creation of a message, agent is not interested in "who and where the receiver agent is", thus messages are created in certain conditions without considering

respondent agents. The composition of messages and the distribution of messages are two separated process. In this node for this level, the agent is provided a message with a name and content to dispatch.

A.3.3. Processing of System Messages

Agents listen to their registered channels, and on the arrival of a message to their channel, they first identify the message, and if the context of the message is relevant to the listener agent then the message is processed. The agent may or may not respond to a message depending on the context of the message. If there is no MEP defined for this message, then it simply ignores the received message.

As it is discussed previously in this chapter, every algorithm has a context. The algorithms related to the MEP's, have a "MEP" string in their context values. As an example MEP-BREAKDOWN, or MEP-OPERATIONSTART-OPERATIONSTOP are sample context strings for the algorithms. In the first case, i.e. MEP-BREAKDOWN, the algorithm handles the messages with a "BREAKDOWN" string in them, and in the latter case, the algorithm will be invoked when the message has a "OPERATIONSTART", or "OPERATIONSTOP" strings in them.

In Figure A.10, a declarative sample MEP algorithm, which is used to handle the BREAKDOWN message in the shop floor, can be seen. First of all as one can see from the first node of the algorithm shown in Figure A.10, one will understand that this algorithm is an algorithm with the context of "MEP-BREAKDOWN" related to a SF_RENEWABLE_RESOURCE_AGENT in the shop floor. The algorithm first finds the related object, i.e. the related machine object in the shop floor, and then activates the state chart SF_RR of this machine object with the BREAKDOWN event, and also activates the SF_RR_DB state chart of this machine object with the UPDATE_DB event. The state chart of the machine object in the shop floor was discussed by discussing the Figure A.2, so activating this state chart with a breakdown event transforms the state of the machine to a BROKEN state with the execution of the DO_Breakdown action, whether it was in an idle, or INUSE state.

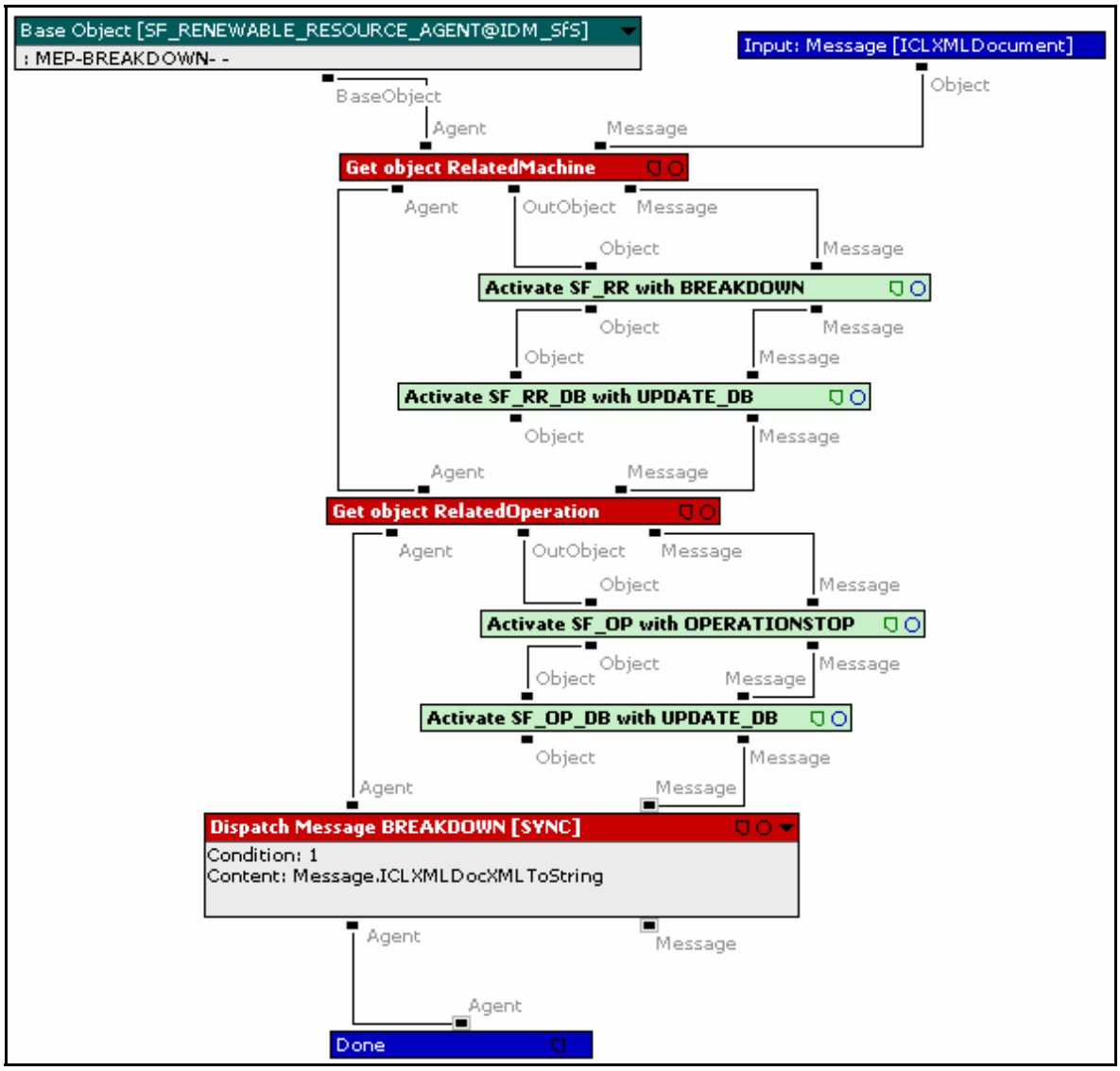

Figure A.10. A sample MEP algorithm for handling of a BREAKDOWN message in the shop floor

Activating the SF_RR_DB state chart shown on the right in Figure A.4, with the UPDATEDB event, transforms the state of database aspect of the machine in the shop floor to a WRITTENtoDB state, i.e. the current status of the object is synchronized with the database. So a BREAKDOWN message is handled by the agent responsible from the shop floor, i.e. the SF_RENEWABLE_RESOURCE_AGENT, and this breakdown message triggered a state change in the related machine object, and furthermore this situation is also synchronized with database with the activation of the SF_RR_DB state chart.

After finishing the execution of these nodes, the MEP now finds the related operation, which was executed on the machine, just before the breakdown. The MEP activates the SF_OP state chart of the related operation object with an "OPERATIONSTOP" event. In Figure A.3 we discussed the state chart of an operation object in the shop floor, and know that this "OPERATIONSTOP" event will make the object a transformation to a STOPPED state by the execution of the DOStop action, if it was in a BEING_PROCESSED state, i.e. if the machine was processing this operation. Finally as in the case of the machine, the current situation of the operation object is synchronized with the database.

In this MEP it should be noted that the received Breakdown message is interpreted as a "BREAKDOWN" event for the shop floor machine object, as "OPERATIONSTOP" event for the operation object, and as "UPDATEDB" event for the database operation of both the operation and machine objects. That shows us a good example of reinterpretation of a message in terms of the related aspects of the related objects. This example shows us that a simple message could be mapped to many events according to the related aspects.

After the processing of the breakdown message for the related objects in shop floor, MEP dispatches the Breakdown message, i.e. it invokes the algorithm which has DISPATCH and BREAKDOWN strings in its context. The MEP further transmits the incoming message without changing it with the help of the dispatch node. It could also be the case where the content may be altered or entered with help of an external algorithm which has a SC_MSG string in its context. The agent dispatches a Breakdown message, without knowing which agents will receive it.

A.4. DISPATCH Implementation in Icron

In order to send a message to an agent, the dispatch node is used. The MEP's define the behaviour of the system according to some events and realizations in the system. In MEP's, the receiver agent decides on certain actions with the help of state charts, and in order to deploy the decisions undertaken, it needs a dispatching mechanism. So actually the message is formed by the agent in a MEP without considering the respondents of the prepared message. The message then will be passed to the dispatch node, and dispatch

node activates the relevant dispatch algorithm within the same context, and it determines the receivers of the message. So the composition of the messages, and the distribution of the messages are isolated from each other, and these are two distinct processes.

Dispatch message node discussed in section A.3.2 is connected to the Dispatch Message Algorithm through a special message handling mechanism implemented in Icron which will be discussed in the coming sections. In a Dispatch Message algorithm there is a special node called Send to Agent Node, for sending a message to another agent specified as the receiver.

A.4.1. Send to Agent Node

In Figure A.11 this special "Send to Agent" node can be seen. The Agent specifies the name and type of the receiver agent, the content of the message and the condition when this message should be dispatched. As it can be seen in Figure A.11 there are three fields to be supplied in order this node will work, which are the following;

- Agent Code: Although it stands for the code of the receiver agent, the Agent Code needs the name of the channel. The message will be submitted to the given channel. If the specified channel has registered sub-channels, then the message will be delivered to all of these sub-channels. Since the agents listen to their respective channels, the message will be received by the agents with the help of these channels.
- Condition: This field is for specifying in which condition the message will be sent.
- Message: This field identifies contents of the message. The expression written here may be used to alter the message text which is created in MEP, or it may be just the supplied message as is. Thus, the content of the message to be transferred is finally determined here.

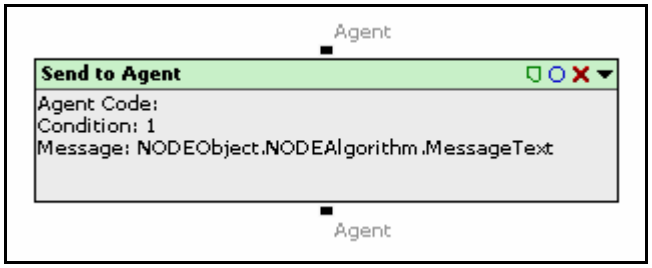

Figure A.11. Send to agent node

A.4.2. Dispatch Message Algorithm

In the case of the MEP's, agents listen to their registered channels, and on the arrival of a message to their channel, they first identify the message, and if the context of the message is relevant to the listener agent then the message is processed with the related algorithm. Almost the same concept is implemented for the dispatch algorithms. In this case the triggering of the identification of the message is done through the dispatch node, rather than the channel listening algorithm. As every algorithm has a context, the dispatch algorithms related to DISPATCH, have a "DISPATCH" string in their context values. As an example DISPATCH-BREAKDOWN-REPAIRCOMPLETION, or DISPATCH-OPERATIONCOMPLETION are sample context strings for the algorithms. In the first case, i.e. DISPATCH-BREAKDOWN-REPAIRCOMPLETION, the algorithm will be invoked when the message has a "BREAKDOWN", or "REPAIRCOMPLETION" strings in them, and in the latter case the algorithm handles the messages with a "OPERATIONCOMPLETION" string in them.

In Figure A.12, a sample dispatch algorithm for handling the "Breakdown" and "Operation Completion" messages can be seen. This dispatch algorithm is actually the algorithm invoked by the execution of the dispatch message node shown in Figure A.10 MEP-BREAKDOWN algorithm. This MEP activated all the relevant state charts in a logical order, and at the end of the algorithm it dispatched the Breakdown message. The dispatch algorithm shown in Figure A.12 is invoked, since it is the algorithm of the agent with a "Breakdown" context. This algorithm sends the same "Breakdown" message with the help of the Send to Agent node, to the "SCHEDULE" channel, and so all the relevant CentralSchedule, JOB, and LS channels (These channels can be seen under the SCHEDULE folder in Figure A.13) will receive and process this message.

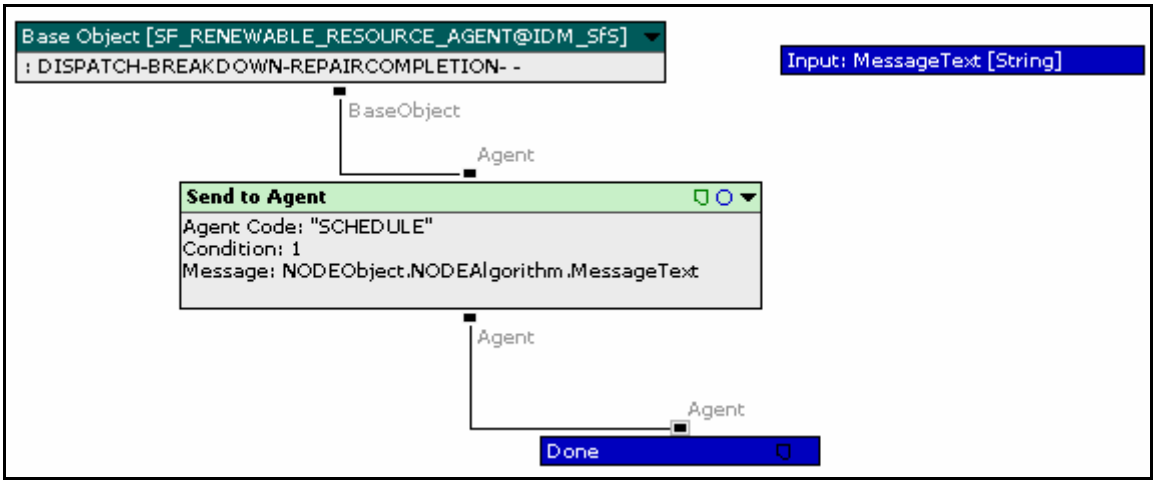

Figure A.12. Sample DISPATCH algorithm for handling the "BREAKDOWN" and "REPAIRCOMPLETION" messages in the shop floor

A.4.3. SC_MSG Algorithm

Like the MEP or DISPATCH algorithms, special type of algorithms with the SC_MSG context is implemented to be able to generate the content of the messages. These algorithms are used especially in the dispatch algorithms' Send To Agent nodes' Message field. As these nodes have the message field to be fulfilled, we prefer to supply the name of an SC_MSG algorithm, so that the separation, readability and reusability of the codes is utilized. These algorithms first generate the XML documents and they convert these XML documents to a string format, as the return values.

A.5. Messaging Mechanism in Icron

In distributed systems, groups of similar or complimentary agents act together to solve problems based on their value systems and communication abilities. The objects, in this case the agents, communicate with each other with the help of a messaging system. So the messaging mechanism is the heart of the overall system, since it enables the agents to coordinate the whole system through communication. The messaging mechanism established in ICRON handles the all the events occurring in the system by building messaging channels between system entities. By using these messaging channels, the system entities communicate with each other and get relevant information with the MEP

implementation, and send appropriate messages in response to the received message with the help of Dispatch nodes.

The channel mechanism implemented in Icron enables both synchronous and asynchronous communication platform. The channels are organized according to the listening agents' base objects. That means the SF_Renewable_Resource agents listen to the channels registered in SF Renewable Resource main channel. All the channels have a registered listening function. When a message is sent to a channel, the listening algorithm, i.e. the generic "Handle Message" algorithm is executed. In Figure A.13, one can see a view of the channels in the development Tab of Icron, and here the Schedule\CentralSchedule, Schedule\JOB, Schedule\SF_RENEWABLE_RESOURCE channels, related to the scheduling messages can be seen. The Schedule\SF_RENEWABLE RESOURCE channel has also a sub-channel called Machine 1, and this Machine 1 channel has a listening algorithm called "Handle Message" on SF_RENEWABLE_RESOURCE AGENT object.

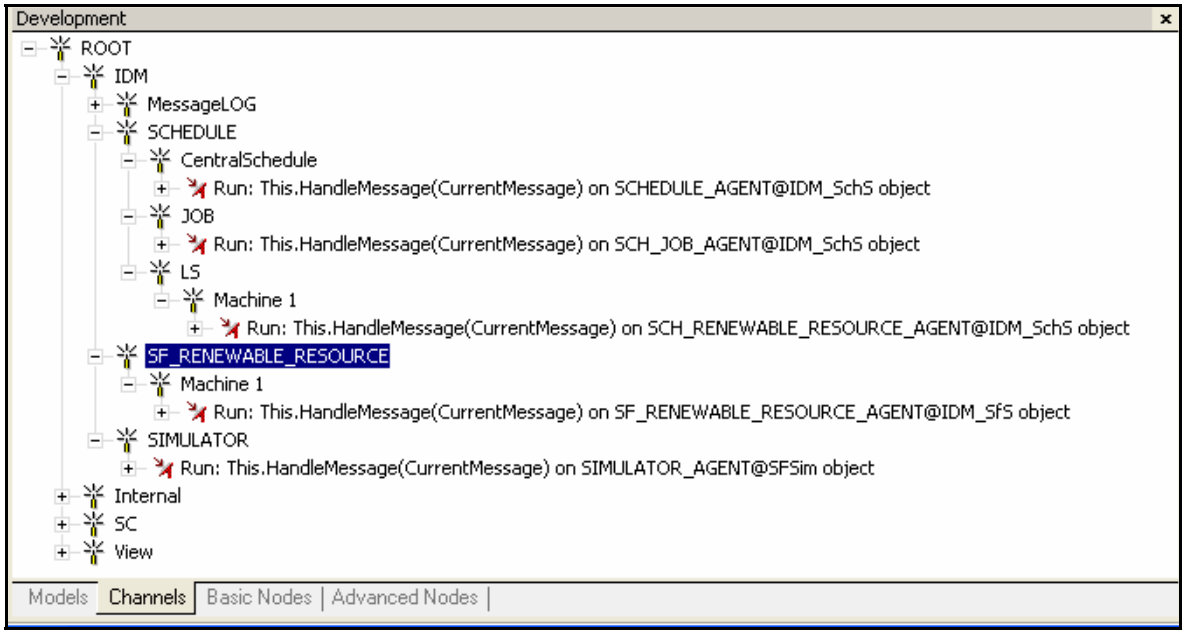

Figure A.13. A view of the channels in the development tab in Icron

As it is discussed earlier, in distributed systems, it may be possible, that the agent and the object to be controlled could be in different domains. Agents which do not exist in the current domain have proxy agents. The functionality of proxy agents are provided by an

addressing mechanism. Each Agent holds either a local address or a remote address. If the receiving agent locally exists in the current domain then it simply receives the message. If the receiving agent does not exist in the current domain then there is a receiver as a proxy of the respondent agent in this domain. The proxy agent simply redirects the message to the remote address it holds transparently. The Figure 3.4 shows the functioning mechanism of proxy agents. The messaging mechanism in Icron uses the TCP/IP protocol for the communication, and the message content is in XML format.

A.5.1. Handle Message Algorithm

The Handle Message algorithm checks the context of the message, whether there is a MEP related with this context. If it finds the relevant MEP algorithm, then it initializes the MEP with the received message, otherwise nothing will be done, and the algorithm will be terminated. In the simplest view, the main objective of this algorithm is to pass the received message to its relevant MEP.

The generic Handle Message algorithm is represented in Figure A.14. The steps of this algorithm follow;

1. The algorithm first checks whether the agent is a local agent, i.e. has a local address or a remote address. If it has a remote address, it redirects the message to the remote address, and the algorithm terminates.
2. The message string is converted to a XML document.
3. The context of the message is checked against the context of all MEP algorithms until the responsible MEP is found.
4. An instance of the MEP algorithm is created and the message is supplied to the MEP algorithm as an input in XML format.

With the help of the Handle Message algorithm the system can process the system messages, by passing them to their relevant algorithms. In this case the relevant messages will be passed to their related MEP algorithms.

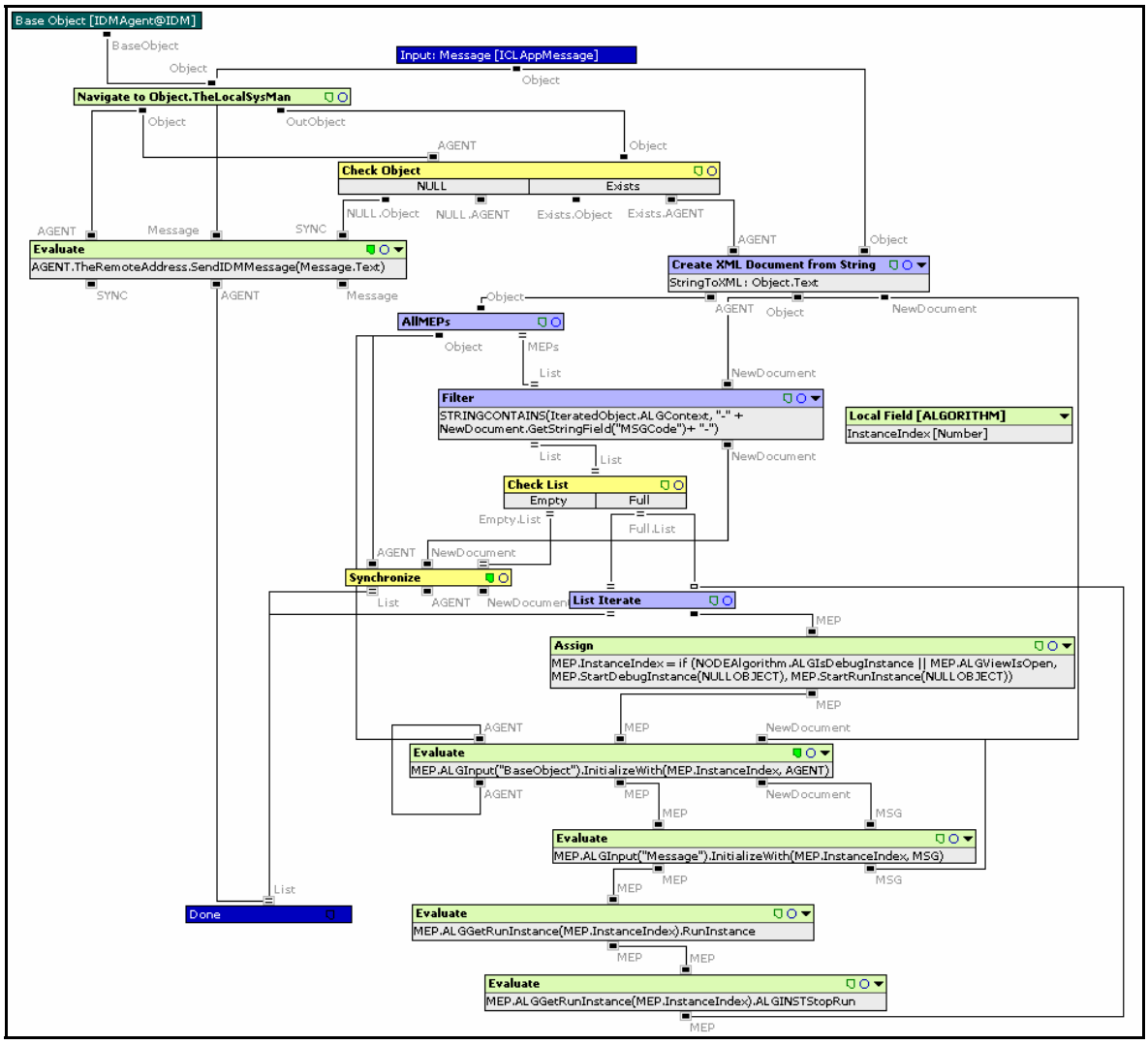

Figure A.14. The generic HandleMessage algorithm for listening the messaging channels

A.5.2. Handle Dispatch Message Algorithm

In the case of the MEP's, the Handle Message algorithms registered to the registered channels trigger the activation of the relevant MEP with the same or similar context of the received message. So in order to implement a generic DISPATCH methodology a similar mechanism is needed. Therefore the triggering of the relevant dispatch algorithm functionality is implemented in this case with the help of the dispatch nodes.

The last node of the MEP-BREAKDOWN algorithm shown in Figure A.10, is actually a dispatch node. The condition of this node is set to 1, which means that the

message will be dispatched in any case with the context BREAKDOWN. Upon the execution of the "Dispatch Message" node, the Dispatch Message algorithm, which is very similar to the Handle Message algorithm in the case of MEP's, is executed. The Handle Message algorithm was used to find the relevant MEP, and now this generic Dispatch Message algorithm is used in the same concept, in order to find the related dispatch algorithms.

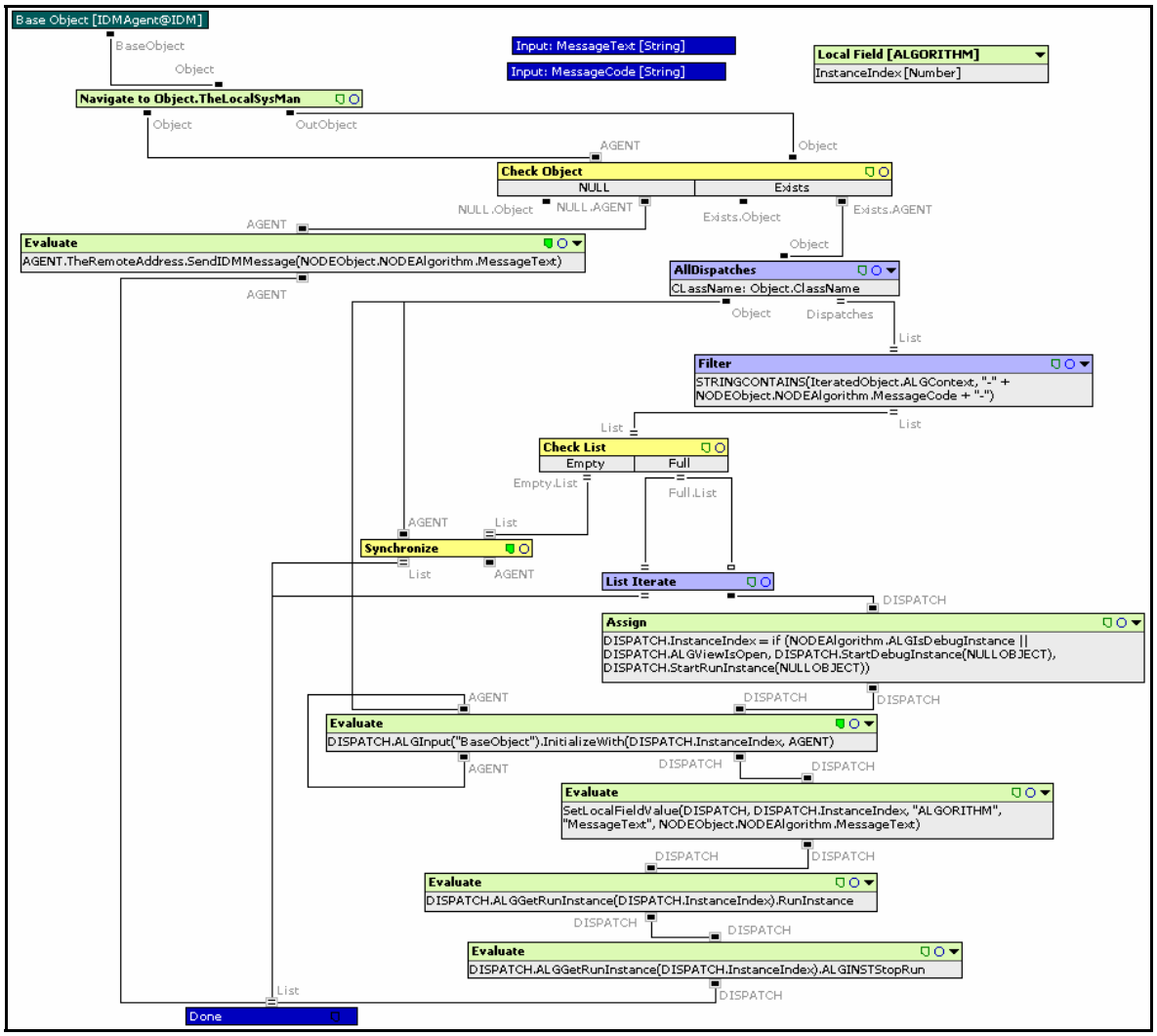

Figure A.15. DispatchMessage algorithm, the relevant dispatch algorithm finder

The DispatchMessage algorithm is represented in Figure A.15, and the steps of this algorithm follow;

1. The algorithm first checks whether the agent is a local agent, i.e. has a local address or a remote address. If it has a remote address, it redirects the message to the remote address, and the algorithm terminates.
2. The context of the message is checked against the context of all DISPATCH algorithms until the responsible DISPATCH is found.
3. An instance of the DISPATCH algorithm is created and the message is supplied to the DISPATCH algorithm as an input.

APPENDIX B. DISTRIBUTED SIMULATOR IN ICRON

In order to be able to work on generic scheduling problems a framework, which can represent and simulate the real life shop floor environments is needed. In real world unexpected events (disruptions) may occur in the shop floor, and they can change the system status and affect the general performance of the system. Vieira et al., 2003, summarized the most common unexpected events in the shop floor as; machine failure, urgent job arrival, job cancellation, due date change, delay in the arrival or shortage of materials, change in job priority, over- or underestimation of process time, and operator absenteeism. These real-time events not only interrupt system operation but also upset the predictive schedule that was previously established. Consequently the resulting schedule may neither be feasible nor nearly optimal anymore.

In dealing with problems associated with these uncertainties, the shop-floor control system plays a very important role. In this chapter distributed simulator of Icron Distributed Model (IDM) is introduced to control and monitor shop floor components and to coordinate the production activities. Simulation is an efficient tool used for modeling, analyzing and design of systems which enables testing different issues of the system.

The Icron Distributed Model (IDM) is composed of a set of distinct systems, communicating through distributed object architecture. Each system models a particular functionality of the enterprise.

Five basic systems are defined in IDM: Physical System (PhyS), Product System (PrdS), Job System (JS), Shop Floor System (SfS) and Scheduling System (SchS). As a general view, dependencies between systems are shown in Figure B.1.

The Physical System is a self-contained representation of the physical organization of the enterprise resources. The resource capacity availability is also defined in the physical system. The *Product System* provides information regarding the product spectrum of the enterprise. The *Job System* is responsible from the management of the collection of customer requests for the products represented in the Product System. *The Shop Floor*

System is responsible for providing information regarding the shop floor status, like inventory levels, breakdowns, acquisition plans, etc. Finally, the *Scheduling System* is the planning engine, which determines how the resources from the Physical System must be utilized to produce the products from the Production System demanded by the customer orders from Job System, based on the constraints imposed by the real current state of the enterprise in the Shop Floor System.

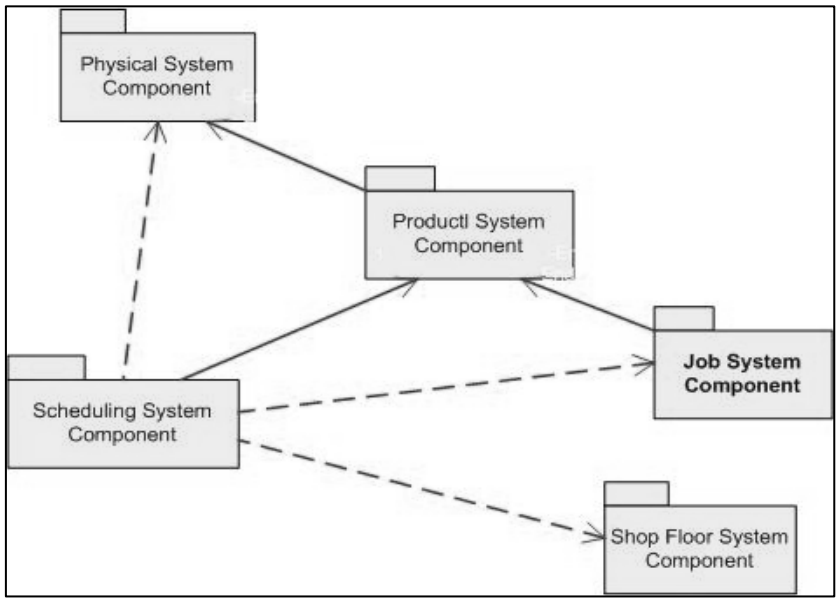

Figure B.1. Overview of the System Components of IDM

After the brief explanation about the systems of Icron, the three basic components of IDM, which are SF Simulator, Shop Floor System, and Scheduling System will be analyzed. SF Simulator is called “SFSim” and represents the renewable resources, i.e. the machines. Shop Floor System is called “IDMSfs” which consists of the real life data of the shop floor environment. Scheduling System is called “IDMSch” and it consists of scheduling agents which are the decision makers for the system.

SFSim: SFSim contains the Event objects, SFSimulator object, SIM renewable resource object and the Simulator Agent, and represents the machines in the shop floor. It holds real life data of the renewable resources in the shop floor system by the SIM renewable resource objects, i.e. machines. To represent a machine like in real life, this object does not know the specifications of the operations it processes. It only knows whether it is processing an operation or not, and if it is processing an operation, it also

knows its current processing speed, the current batch size, and the start time for the operation. The machine object also knows whether it is broken or not, and when it is broken then it also knows the time of the breakdown, and the estimated repair duration. So it has enough information to determine its current state, from the set of states represented in the SIM_RR state chart shown in Figure A.6.

The ultimate goal was to represent a real life machine with the help of an object structure. Whether a highly complicated CNC machine or a very simple turret lathe is represented, one knows that these machines either process an operation, are idle, or broken. When it processes an operation one also knows the operation start time, the batch size to be processed on this machine also. Therefore these SIM renewable resource objects will be the counterparts of the real life machines, and they simulate the behaviour of the real life machines. In other words, when a real life machine with communication intelligence is replaced with these objects, then instead of the simulator triggers the messages, the messages can be received even from these real manufacturing entities.

Shop Floor System / IDMSfS: IDMSfs is the software component of a shop floor control system such as machine drivers or controllers. It is able to represent the real life data with the execution data coming from the software applications. Renewable resources in a shop floor are physical entities that can provide very basic information about their situations i.e. current status information of machines. These kinds of software components combine the real life data of machines with the data of other scheduling entities of shop floor environment. For example IDMSfs includes SF Renewable Resource objects to represent the machines and SF_OPERATION objects to represent the operations. The SF_OPERATION objects hold the dynamic shop floor execution data about each operation in the system such as batch size, the current resource, the last operation start time, percent completion of last unit, realized completion time etc. If a machine, i.e. SF Renewable Resource, is currently processing an operation, then with the help of the "TheCurrentOperation" pointer, it has access to its current operation object.

Scheduling System / IDMSchS: IDMSchS consists of SCHEDULE_AGENT's which act as global schedulers, a SCH_JOB_AGENT which manages predecessor - successor relations of operations, and SCH_RENEWABLE_RESOURCE_AGENT's

which act as local schedulers. These agents make scheduling decisions by their individual computing facilities based on their local information and local response and scheduling policies.

In summary, the simulator generates and holds all the realizations, and provides these realizations to the shop floor system, i.e. IDMSfS, which represents the execution of the shop floor. The shop floor execution information is stored in a central database, and the realizations in the shop floor are synchronized to this database. The scheduling agents, i.e. the decision makers, gather information about their problem instances by querying this database, and take part in the simulation process by dispatching the schedule orders to the relevant objects.

B.6. Events and Event Handling in ICRON

There is a discrete time simulation engine in Icron. With the help of this simulation engine one can create a set of events with a realization date. The simulation engine generates simulation time steps, and according to the interval defined for this time steps, the simulation engine generates a timestamp corresponding to real life time. So the definition of the interval for each simulation step acts like a time magnification factor. Let's say that each simulation step occurs each second, so if one defines one day for each step, then only one second in the simulation time will represent a whole day in real life. Thus the simulation system advances with constant simulation time blocks, and the set of events generated with a timestamp less than or equal to the current time of the simulator, these events will be assumed as realized.

As the whole system is composed of objects, the SFSim has an Event class, and all specific type of events are extended from this event class. The following events are defined in our simulator;

1. *Breakdown event:* The breakdown event is created in order to simulate real life breakdowns of machines. The event is generated according to the guidelines defined in section 6.4.5.

2. *New Job event:* The new job is generated to simulate arrival of a new job in real life. This event is generated according to the guidelines defined in section 6.4.2.

3. *Repair Completion Event:* This event will be fired when the broken machine is repaired, and is ready to be used. This event is generated according to the guidelines defined in section 6.4.5.

4. *Operation Completion Event:* This event will be fired when an operation is completed its processing on a machine. The operation completion event will be related with the processing time distribution.

5. *Delta Event:* This event is used in order to implement the decision time effect of the system. In our system each scheduling agent two schedules, which are active and passive schedules. The agent runs the scheduling operation on its passive schedule, and the resulting schedule will be released a delta time interval later in order to represent the effect of the decision time. When this event is fired the prepared schedule is released. Thus one can think delta event mechanism as a reminder mechanism, which reminds one at a predetermined future time.

    a. *Central Schedule (CS) Event:* Central schedule event can be seen as a sub event. With the help of the Delta Event it is used to trigger a cyclic response policy for global scheduling agents. The name is obsolete, but since the implementation is valid for this system, it is left as is.

The simulator arbitrarily generates the simulation events including breakdown event of machines, new job arrival event and central schedule event in order to symbolize changing situations. When a breakdown event is generated the repair duration of the related machine is also determined and the related repair completion event is placed into the events queue of the simulator. Similarly, when a machine starts to process an operation with respect to the direction of a dispatcher, the corresponding operation completion time is calculated with the help of the processing time of this operation on this machine and the estimated completion time is placed as an operation completion event into the events queue of the

simulator. Finally when a scheduling agent begins the scheduling process it also places the estimated release time of this schedule as the delta event into the simulators event queue.

B.6.1. Processing of Events

All the event types are extended from the very same event class. The main event class has a generic Handle method. All the event types, which are extensions to the main event class has their own Handle method. The processing of events is performed with this Handle Message algorithm very similar to the implementation of the MEP and DISPATCH message handling algorithms. When a new event is fired by the simulation engine, the "Handle" algorithm of the main event class is invoked. This generic "Handle" message algorithm which can be seen in Figure B.2.

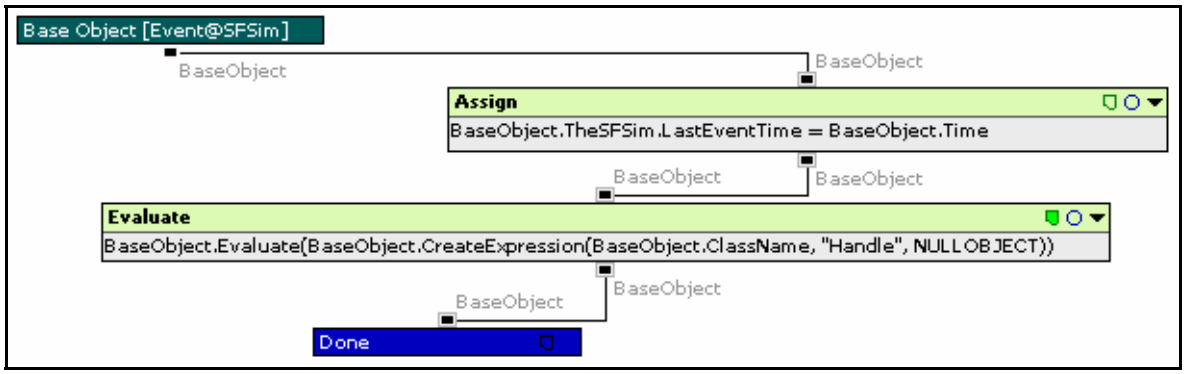

Figure B.2. The "Handle" algorithm of the main event class

The steps of this algorithm follow;

1. The algorithm assigns the Last Event Time attribute of the SFSimulator object to the current time.
2. It evaluates the event, and finds the base object of the Event type arrived, and invokes the "Handle" algorithm of the base object. Thus if the event is a Breakdown event than this node invokes the Handle method of the Event_Breakdown object. This will be the same for all the other event types defined previously.

The events form one of the most important parts of the simulation. Therefore for the sake of completeness all the specific Event Handle algorithms will be discussed.

B.6.1.1. Handle Algorithm of the Breakdown Event: This algorithm first calculates the estimated breakdown duration. Then it creates a new Breakdown message with MSGCode, Event Time, Machine Code, and the Estimated Breakdown Duration fields. The first field has the "BREAKDOWN" string as its value, and the remaining fields represent the current time, the object code, and the calculated estimated breakdown duration.

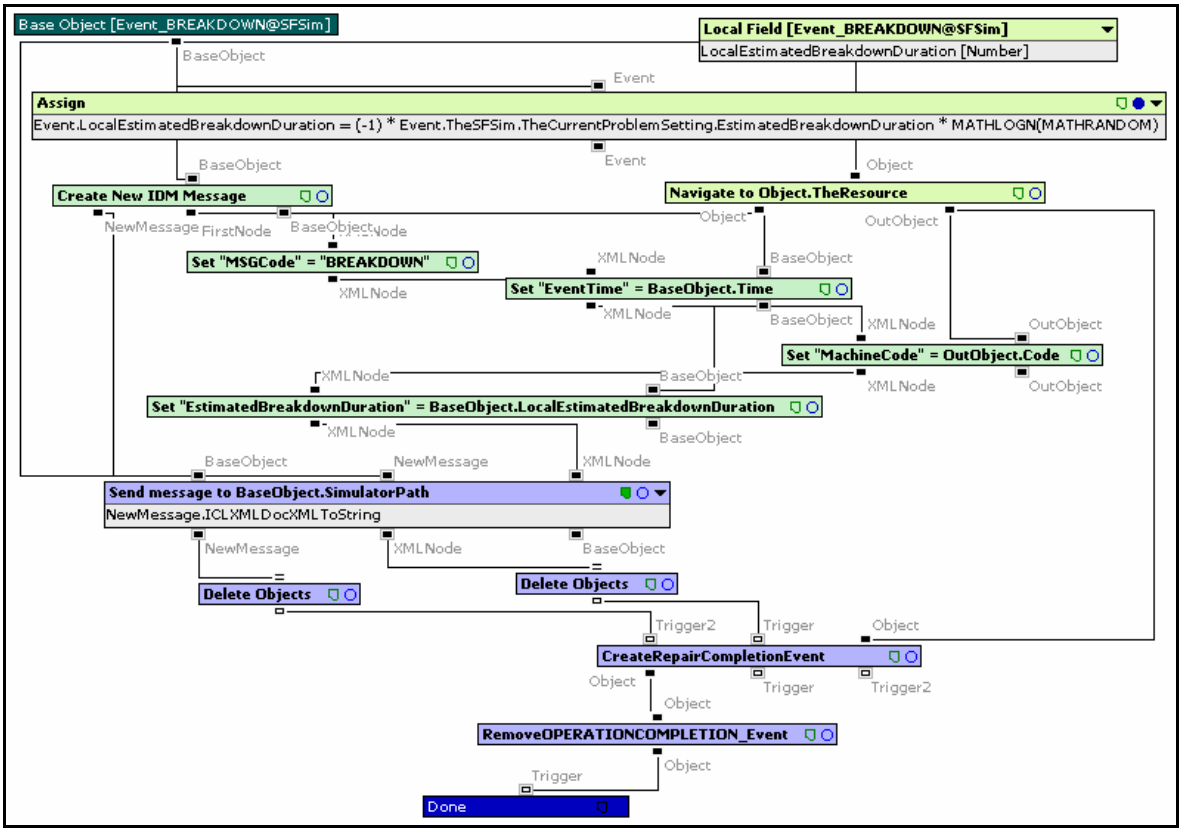

Figure B.3. Handle Algorithm of the Event BREAKDOWN

The message is sent to the Simulator Agent using the Simulator channel. A repair completion event for this breakdown is created. Finally the previously generated operation completion event will be deleted from the simulators event queue. If the machine was in an idle state, i.e. it was not processing any operation, before the arrival of the breakdown

event, then the last RemoveOperationCompletionEvent node will not delete anything from the event queue, since there is no such event registered to the queue.

B.6.1.2. Handle Algorithm of the New Job Event: This algorithm is the most complex event handling algorithm shown in Figure B.4.

Although New Job event handler algorithm seems quite complex, it does only prepare a message with all relevant properties for defining a new job, with the message code field having the "NEWJOB" string, and sends it back to the simulator agent, and creates a new job event.

The new job message contains the values for the following parameters; MSGCode, Event Time, Job System, Job Code, Order Code, Product System Code, Part Code, Routing System Code, Routing Code, Routing Version, nBatch, Due Date, and Release Time. The MSGCode stands as an identifier of the message, as the name implies, it represents the message code. The time of the event is represented by the Event Time field. The Job system and Job code represent the job specific parameters. Order Code, Products System Code, Part Code, Routing System Code, Routing Code, and Routing version represents the order, part, and routing specific parameters. The batch size is represented with the nBatch parameter. Due date as the name implies represents the due date of the job, whereas the Release Time stands for the release time of the job.

A new job arrival to a system is not dependent with the properties of the shop floor, therefore an external triggering mechanism should be there in order to have continuous new job arrivals. Therefore in the very beginning, i.e. in the initialization step, a special initialization algorithm creates the very first new job event. Upon finishing the execution of the Handle algorithm of this event, the algorithm creates a new "New Job" event, so that the system gets continuous new job arrivals.

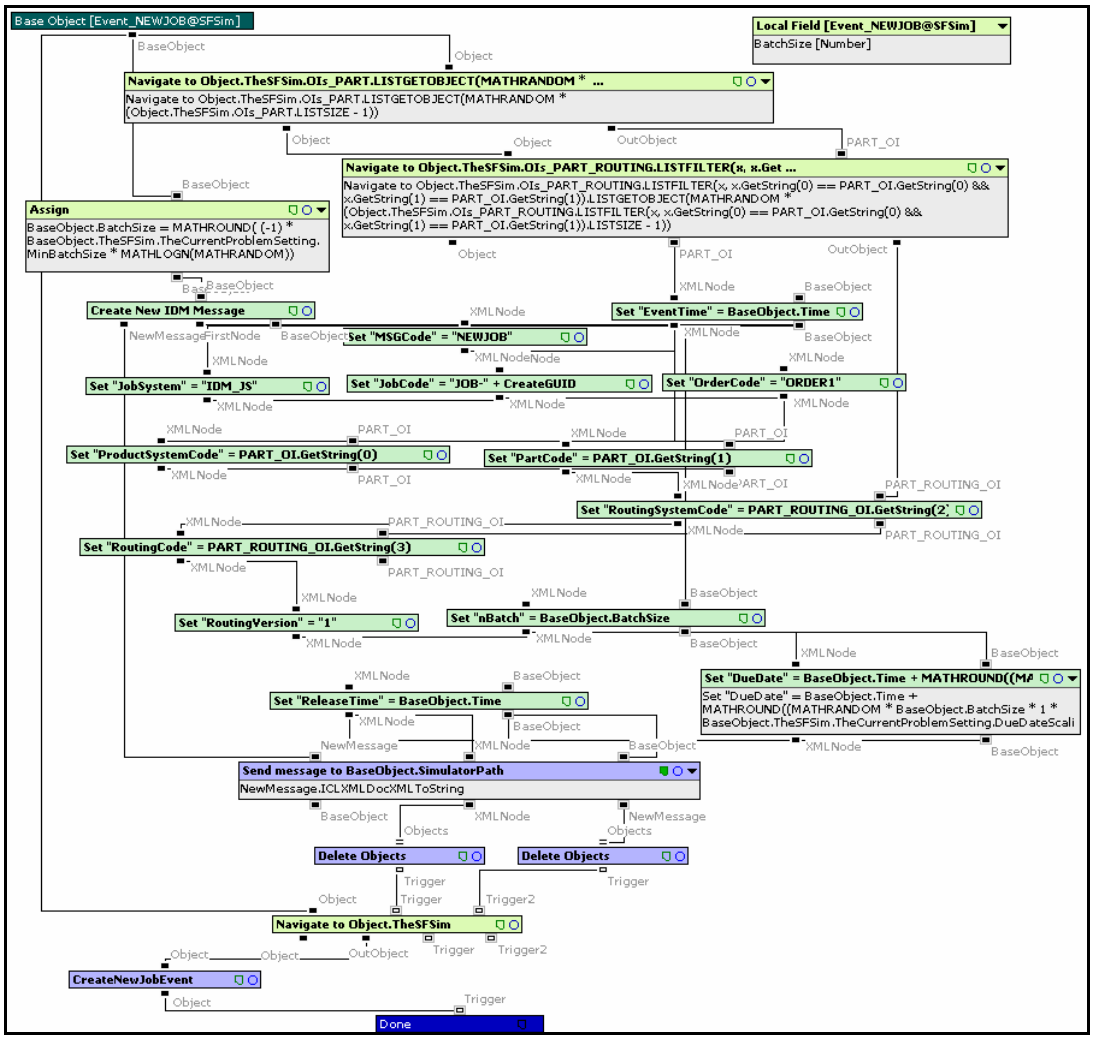

Figure B.4. Handle algorithm of the event NEWJOB

B.6.1.3. Handle Algorithm of the Repair Completion Event: Repair Completion Event's "Handle" algorithm can be seen in Figure B.5. This algorithm first creates a message with the message code field having the "REPAIRCOMPLETION" string. The Repair Completion message contains MSGCode, EventTime, and MachineCode fields. The event time represents the simulator time and MachineCode represents the code of resource object, which holds a pointer to a SIM renewable resource object, i.e. it sends the code of the machine which is currently operating the related operation.

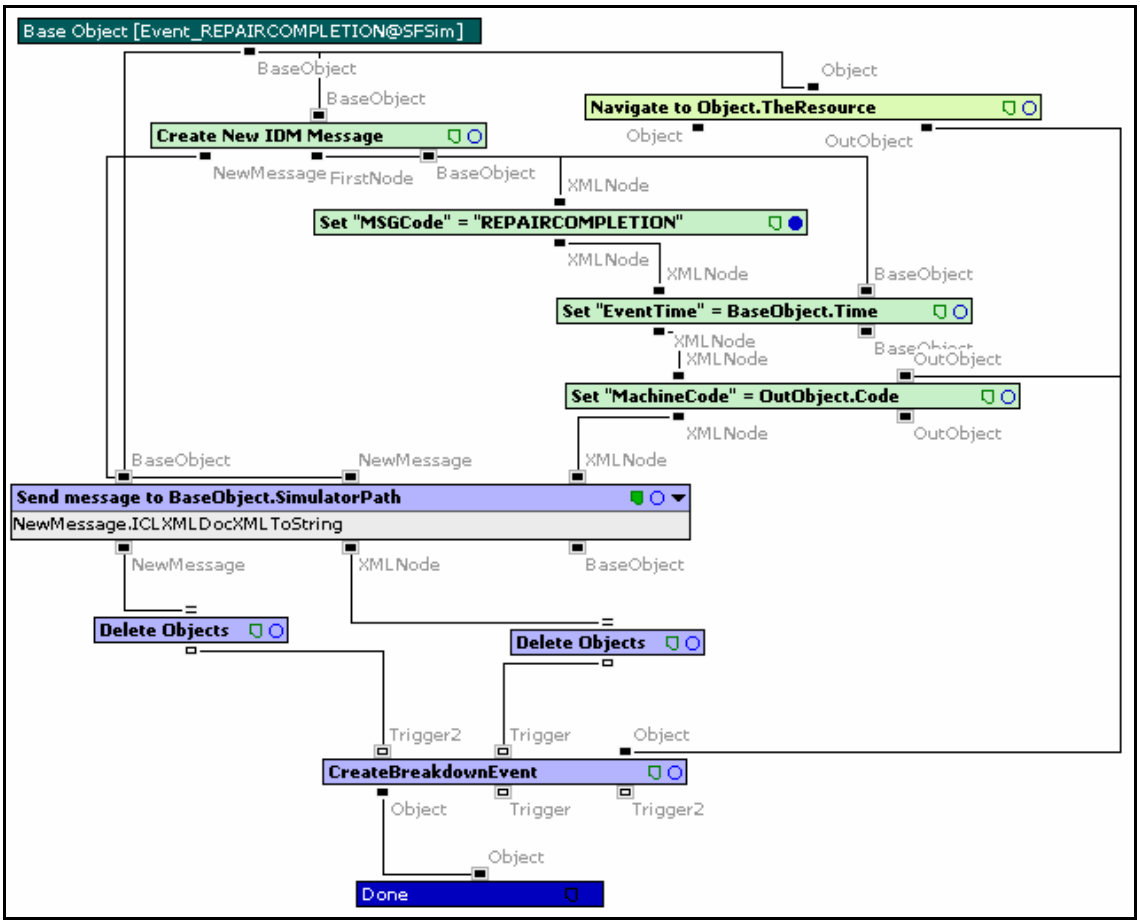

Figure B.5. Handle algorithm of the event REPAIRCOMPLETION

After sending this message to the simulator agent with the help of the simulator channel, the algorithm creates a new breakdown event for this machine object. As it is discussed previously, in the case of the handle algorithm of the Breakdown event, the algorithm generated a repair completion event, and now this repair completion event algorithm creates a breakdown in return.

B.6.1.4. Handle Algorithm of the Operation Completion Event: This Operation Completion Event's "Handle" algorithm can be seen in Figure B.6.

This algorithm first creates a message with the message code field having the "OPERATIONCOMPLETION" string. As in the case of the repair completion message, the operation completion message also contains the EventTime, and MachineCode fields besides the MSGCode field. With the help of these fields, the message contains the

simulator time and the code of resource object, which holds a pointer to a SIM renewable resource object, i.e. it sends the code of the machine which is currently operating the related operation. Finally it sends this message to the simulator agent with the help of the simulator channel.

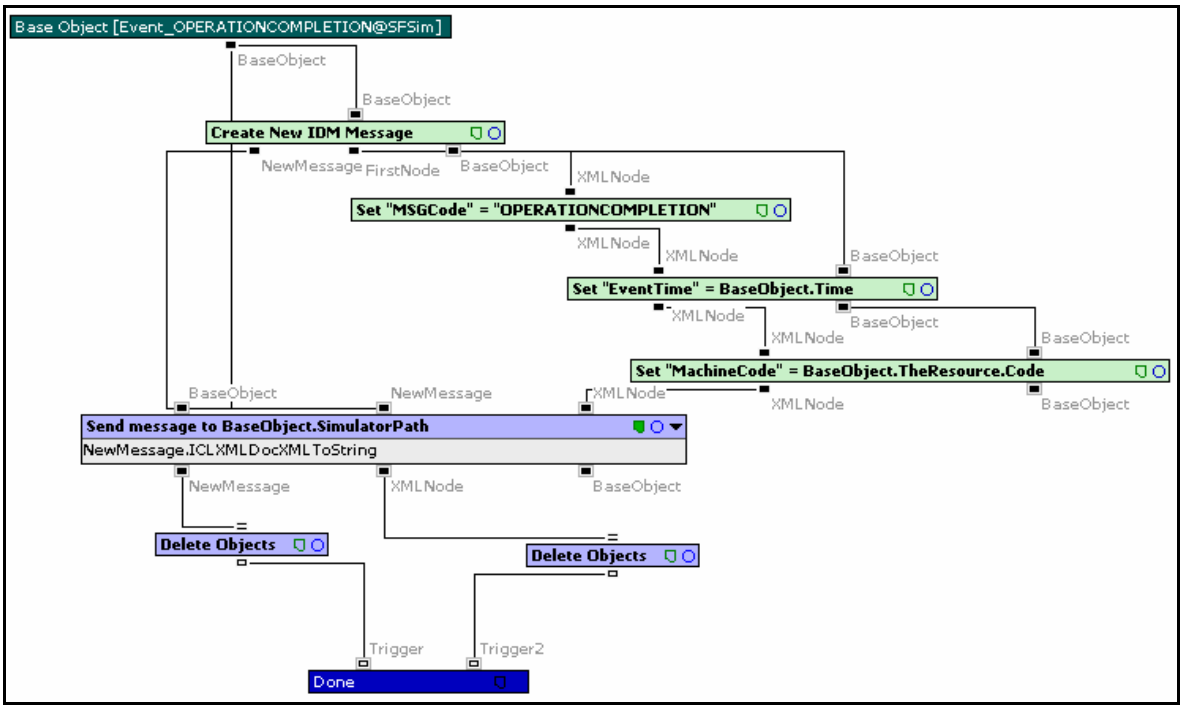

Figure B.6. Handle Algorithm of the Event OPERATIONCOMPLETION

B.6.1.5. Handle Algorithm of the Delta Event: Delta Event's "Handle" algorithm can be seen in Figure B.7. This algorithm first creates a message with the message code field having the "DELTA_MSG" string. The message also contains the simulator time, and two important fields, namely the return message, and the return path information. It sends this message to the simulator agent with the help of the simulator channel. This Delta Event Handle algorithm is a special algorithm with a callback function implementation. The message contains the callback functions return message, and the return path. A sample usage of this special event can be seen in the MEP-CS algorithm shown in Figure B.9.

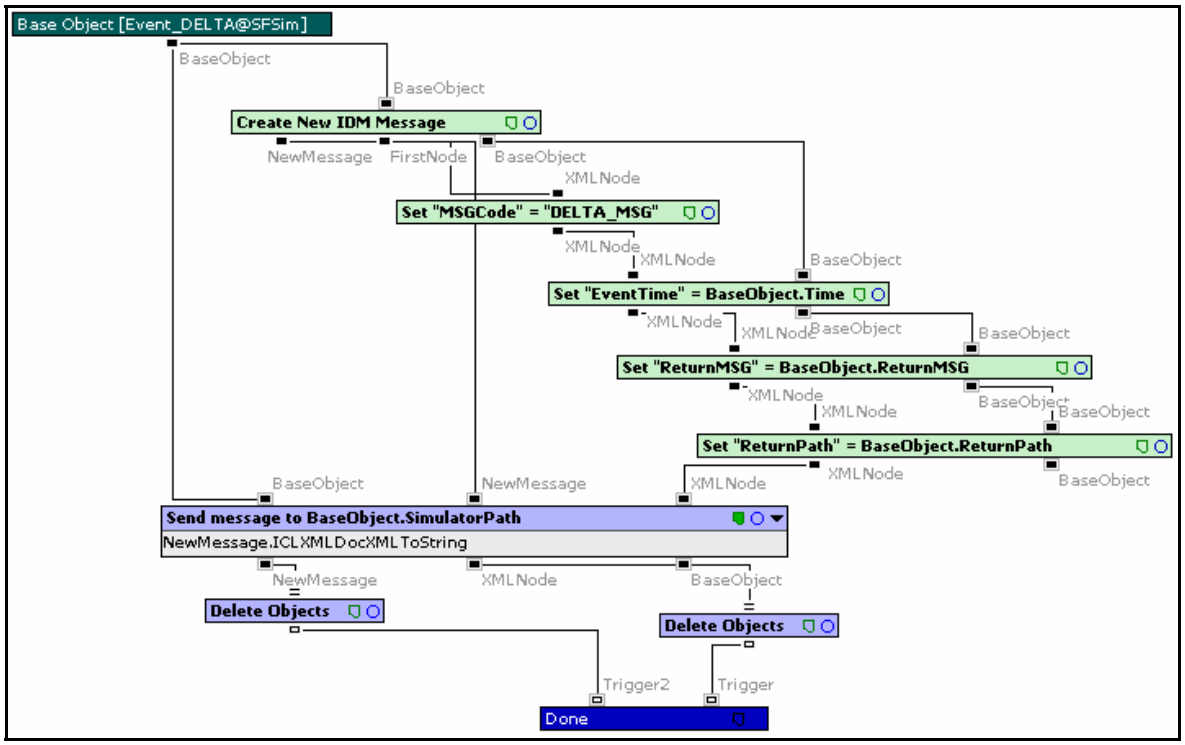

Figure B.7. Handle Algorithm of the Event Delta

The MEP-CS algorithm represents one of the decision points implemented in this study. According to the control policy dedicated by the problem setup this algorithm checks whether the job system agent stability is stable or unstable, whether the decision process takes time or not, and finally whether there is a cyclic or event based response policy. So when the agent stability is set to unstable, and the schedule agent is currently in a scheduling state, then the agent simply ignores the incoming message. Otherwise it gets the passive schedule, and synchronizes its time, and synchronizes with the database, and starts to schedule on the passive schedule. If the scheduling process, i.e. decision process takes time, it dispatches the REGISTER_DELTA_TIME message, with the help of Delta Message algorithm shown in Figure B.8, and with the CS_DELTA as delta message code, and RELEASE_CS event as the Return message code parameters. The algorithm will end by dispatching another REGISTER_DELTA_TIME message, with CS_CYCLE and CS parameters in the case of a cyclic response policy in order to ensure the creation of new CS events.

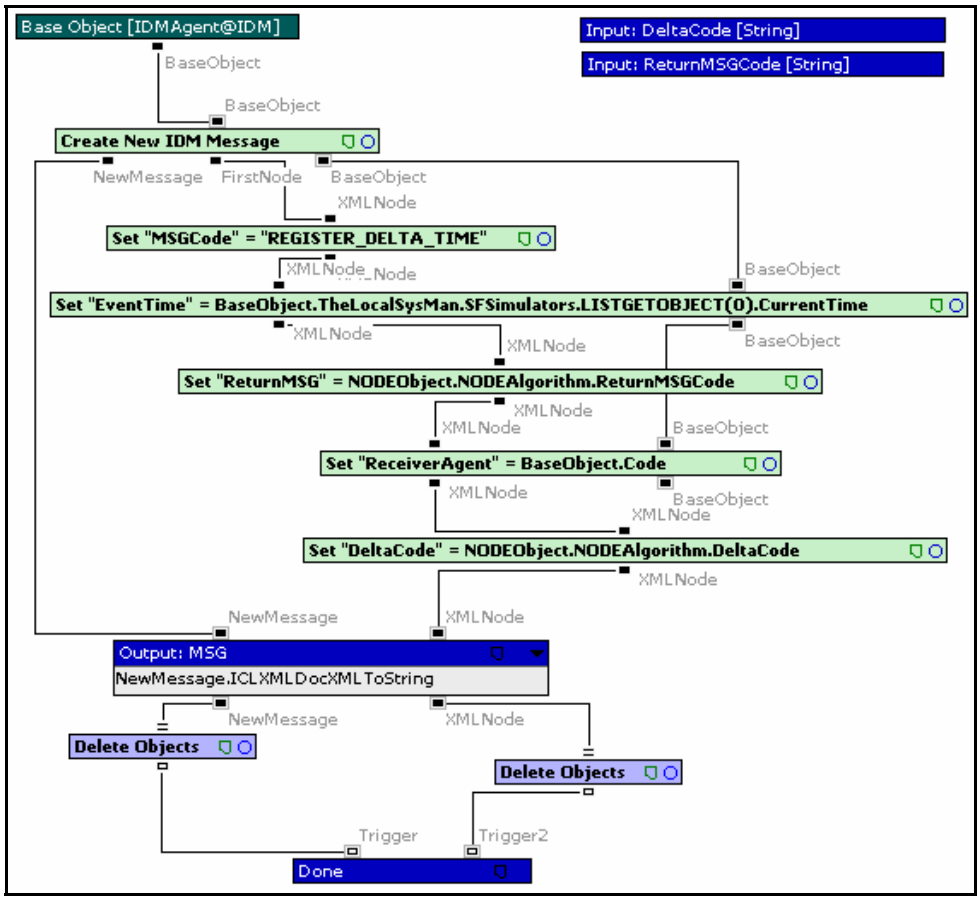

Figure B.8. DeltaMessage algorithm

The Delta Message algorithm creates a message with message code set to REGISTER_DELTA_TIME. It adds the simulators current time as event time, the return message, the receiver agent and the delta code to the formed message. This formed message will be sent to the simulator agent. Upon receiving this message the simulator agent will handle this message with its MEP-REGISTER_DELTA_TIME algorithm, and will first find the duration time for the given Delta Code, and will fire an event, which is defined by the return message code, i.e. in this case as shown in Figure B.9, a RELEASE_CS event, to be received by the agent which registered this delta time. So in this case a delta time later, the schedule agent will receive this RELEASE_CS event.

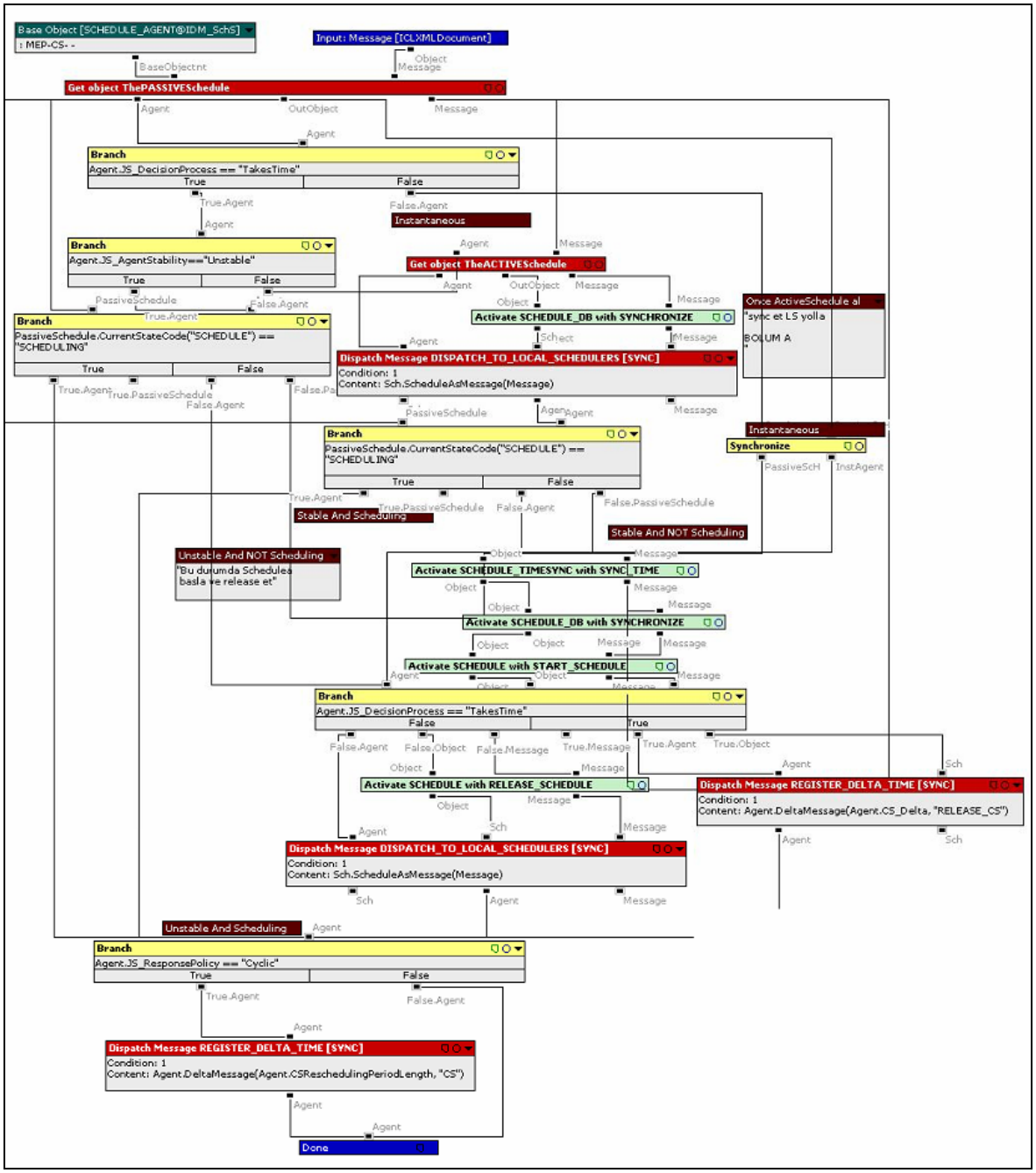

Figure B.9. A sample MEP algorithm for handling of a CS message

By examining the MEP-CS algorithm one can see, if the decision process is instantaneous, then the schedule agent will immediately release the schedule, and dispatch the new schedule to local schedulers with the help of another important algorithm called ScheduleAsMessage. The SCheduleAsMessage algorithm will create an XML document which contains the schedule. Again according to the agents' response policy it will either dispatch another CS event in the case of a cyclic response policy as a

REGISTER_DELTA_TIME message or will simply end the algorithm in the case of an event based response policy.

B.7. Messaging Flow of the Simulation Events in ICRON

B.7.1. Messaging Flow for Breakdown simulation Event

In this section the messaging flow for the Breakdown event will be analyzed in detail. The messaging flow of a breakdown event can be seen in Figure B.10. The message which is created by the simulator will be first received by the simulator agent. Then it will be sent to the SF Renewable Resource agent, and from there it will be broadcasted to the scheduling agents.

When the simulator engine fires a breakdown event, i.e. it sends a message with a message code as BREAKDOWN to the simulator agents listening channel, the message will be received by the simulator agent's corresponding MEP with the help of the generic Handle Message algorithm which is previously discussed in section A.5.1. By receiving this BREAKDOWN message the Simulator Agent will trigger the MEP_BREAKDOWN which is shown in Figure B.11.

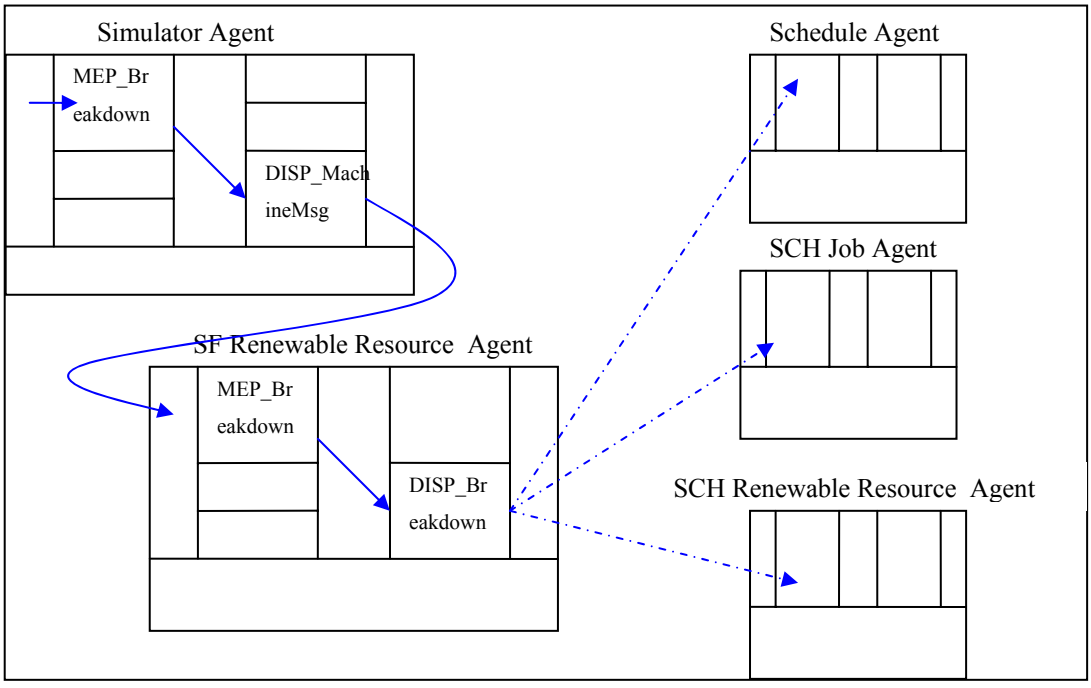

Figure B.10. Breakdown event message flow diagram

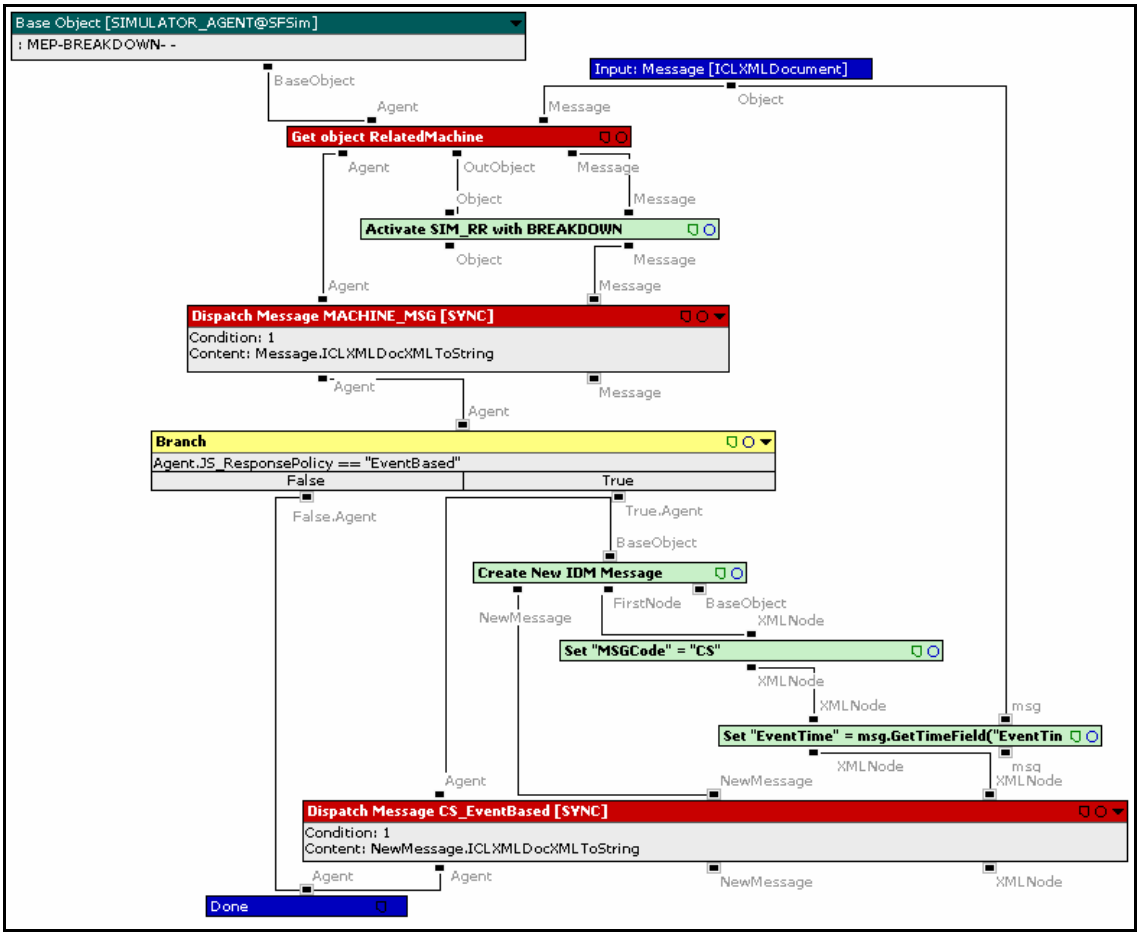

Figure B.11. MEP-BREAKDOWN algorithm for Simulator Agent

The MEP-BREAKDOWN will first activate the SIM_RR state chart of the SIM renewable resource object with the BREAKDOWN event. After the state chart activation, the breakdown message is dispatched with the help of DISPATCH-MACHINE_MSG algorithm, which can be seen in Figure B.12. When the agent response policy is defined as event based in the control policy parameters, then this control policy dictates that the system will try to respond to every event. To be able to respond to every event we introduce a new CS event at the end of handling of these events in related agents. In this case we introduced this CS message in the MEP-BREAKDOWN of the simulator agent. In order to be able to implement the event based response policy, we defined the MEP-Breakdown, MEP-OperationCompletion, MEP-RepairCompletion of the simulator agent, and MEP-NewJob, MEP-CS of the schedule agent as decision points.

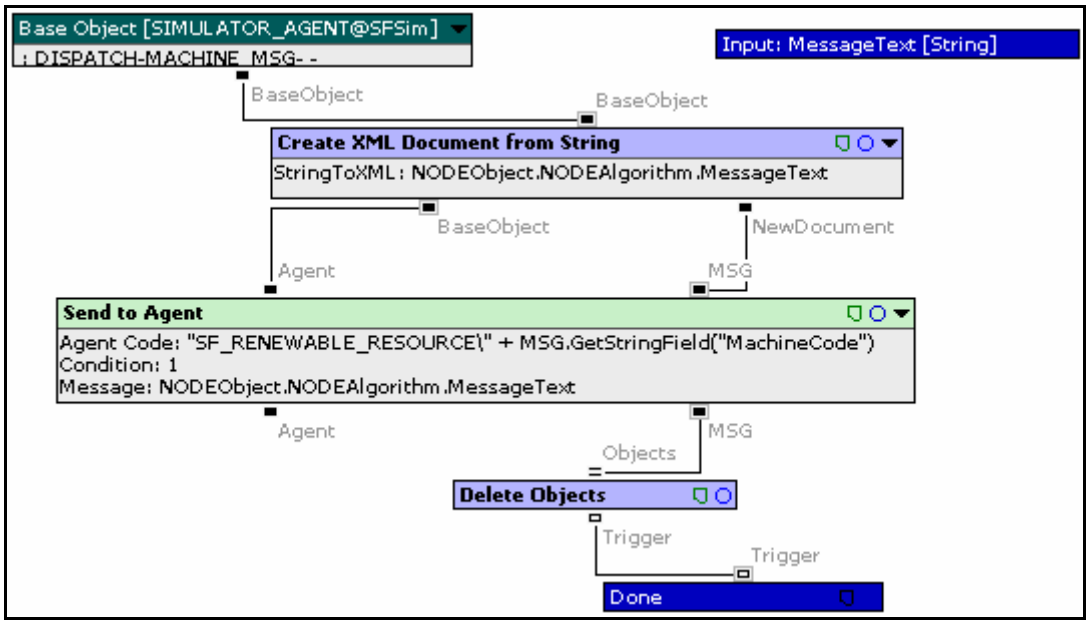

Figure B.12. DISPATCH_Machine_MSG algorithm

The dispatch algorithm will first create an XML document from the string in order to extract the machine code field from the message. The Send to Agent node will send the prepared message to the shop floor machine object sub-channel under the SF Renewable Resource channel, with the help of this extracted machine code value. Then the sent message will be received by the SF_RENEWABLE_ RESOURCE_AGENT's relevant MEP algorithm, which will be in this case the MEP-BREAKDOWN of SF_RENEWABLE RESOURCE_AGENT.

This MEP_Breakdown algorithm first activates the SF_RR state chart with the event breakdown, i.e. changes the corresponding shop floor renewable resource objects state. Then activates the SF_RR_DB state chart with the UPDATE_DB event, i.e. it synchronizes the current status of the corresponding shop floor machine to the database. After that the current operation processing on the same machine will be handled, if any. So the SF_OP state chart of the related operation object will be activated with the OPERATIONSTOP event, and again with the activation of the SF_OP_DB state chart with the UPDATE_DB event, the operations will be synchronized with the database. Finally the BREAKDOWN message will be dispatched with the help of the dispatch algorithm shown in Figure B.14.

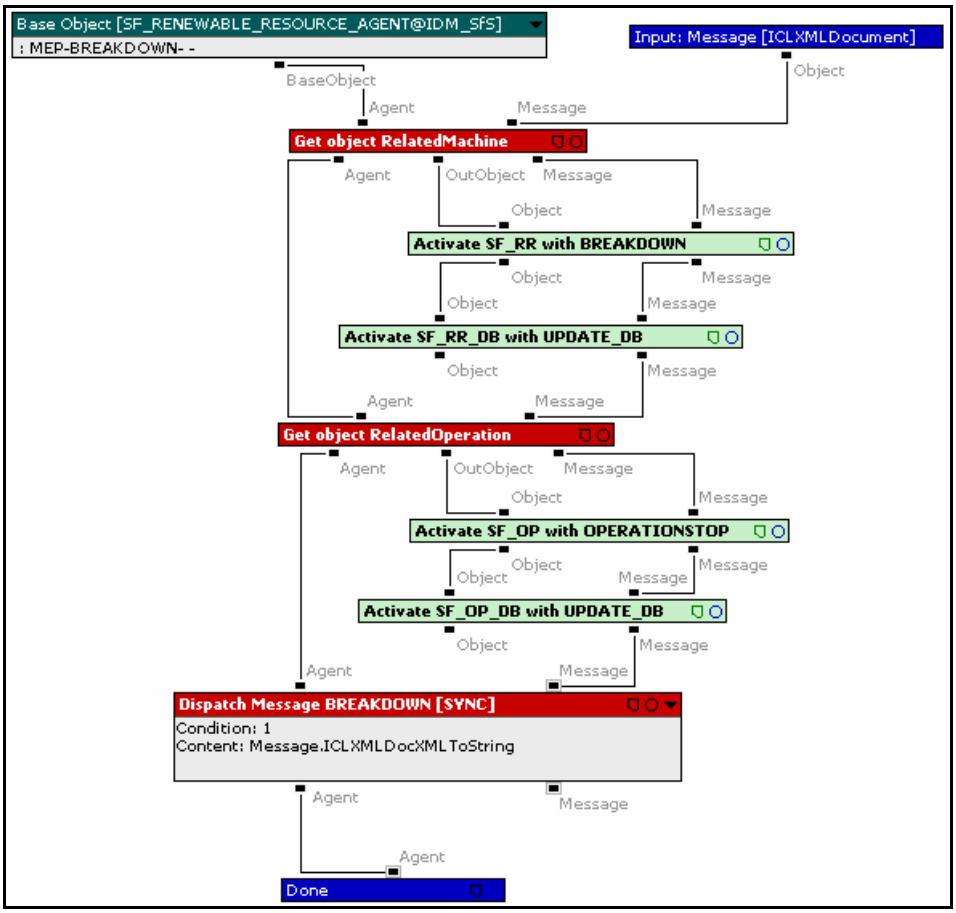

Figure B.13. MEP_BREAKDOWN algorithm in the SF_RENEWABLE RESOURCE_AGENT

The dispatch algorithm will send the same message to the SCHEDULE channel, so every agent, which is connected to a subchannel of this main channel will receive this message.

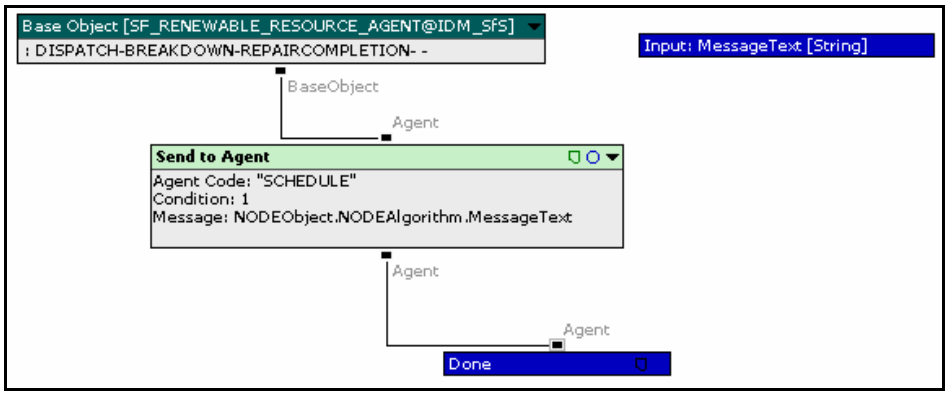

Figure B.14. SF Renewable Resource DISPATCH Breakdown algorithm

B.7.2. Messaging Flow for New Job Event

In this section the messaging flow for the New Job event will be analyzed in detail. The messaging flow of a new job event can be seen in Figure B.15. The New Job event is created by the simulator, and it is received by the simulator agent. It then dispatches the new job message to all scheduling agents, i.e. to SCH Job Agent, Schedule Agent, and finally SCH Renewable Resource agents.

When the simulator engine fires a new job event, i.e. it sends a message with a context string NEWJOB to the simulator agents listening channel, the message will be received by the simulator agent's corresponding MEP. By receiving this NEWJOB message the SimulatorAgent will trigger the MEP_NEWJOB which is shown in Figure B.16.

The MEP-NEWJOB will dispatch the received message with the help of DISPATCH-NEWJOB_MSG algorithm, which can be seen in Figure B.17.

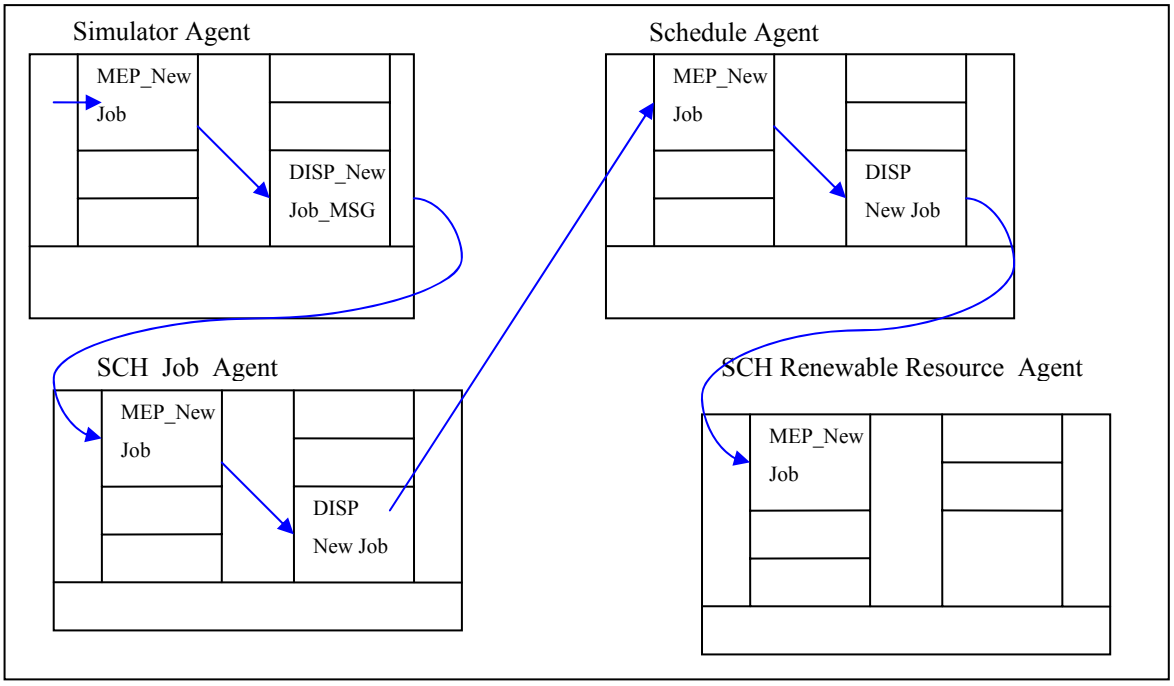

Figure B.15. New Job event message flow diagram

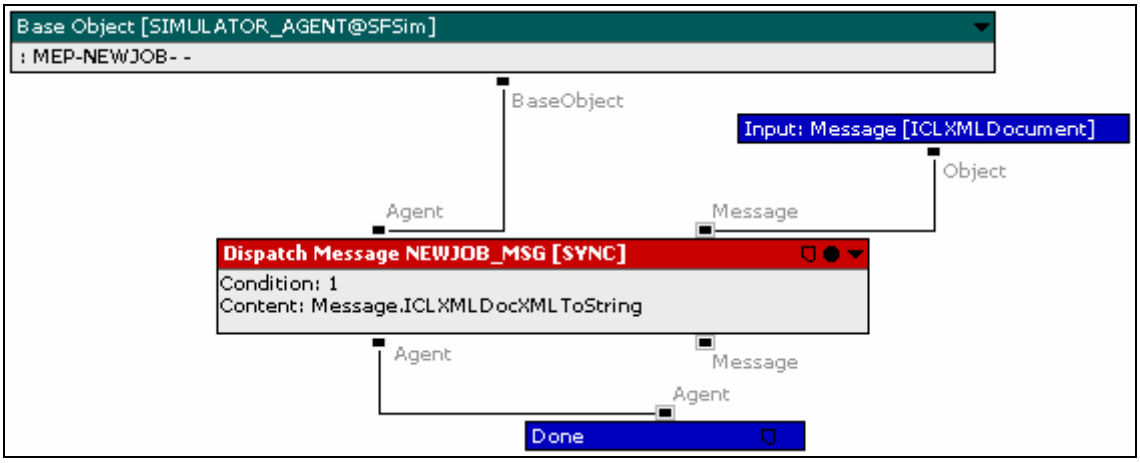

Figure B.16. MEP-NEWJOB algorithm for Simulator Agent

The dispatch algorithm initiates Send to Agent node to send the message to the job system object sub-channel under the SCHEDULE channel. Then the sent message will be received by the SCH_JOB_AGENT's relevant MEP algorithm, which will be in this case the MEP-NEWJOB of SCH_JOB_AGENT.

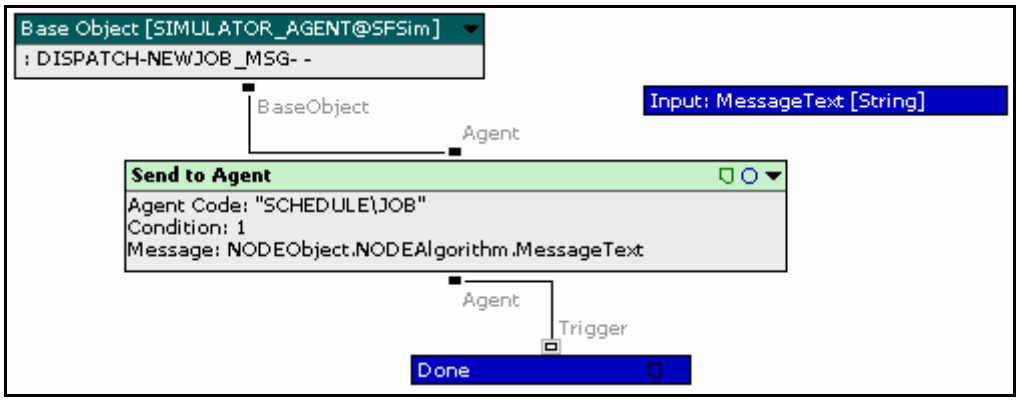

Figure B.17. DISPATCH_NEWJOB_MSG algorithm

The MEP_NEWJOB algorithm first activates the SCHEDULE_JD state chart with the event Create_NEWJOB. The Job dispatcher state chart is a special state chart which has a DONE state, and which is always true. That means this object will be always stable, and therefore is ready to process incoming requests every time. A new job will be created in the job systems schedule after the execution of a CreateNewJob action triggered in this state chart. The MEP then will activate the SCH_JOB state chart with the RELEASE event to inform the shop floor system database from the incoming new job. After that the

SCHEDULE_DB state chart is activated with the synchronize event. The shop floor objects, i.e. machines will be aware of the incoming new job, and the shop floor is synchronized with database. Finally the NEWJOB message is dispatched. It will trigger the DISPATCH-NEWJOB algorithm shown in Figure B.17.

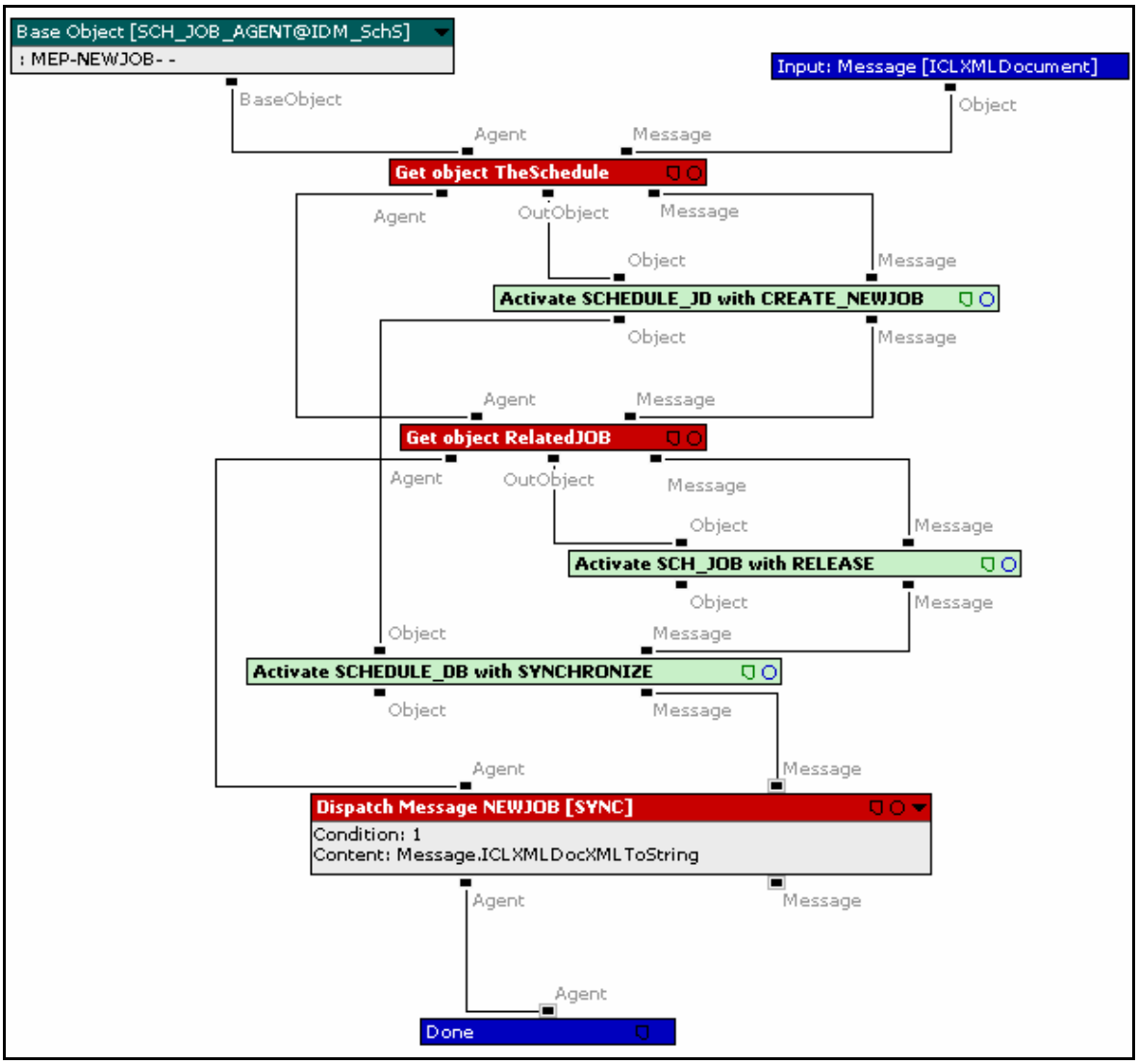

Figure B.18. MEP-NEWJOB algorithm in the SCH_JOB_AGENT

The dispatch algorithm will send the same message to the Schedule agent, i.e. to the SCHEDULE\CentralSchedule channel. This message will be received by the MEP-NEWJOB algorithm of the Schedule Agent. As in the case of the MEP-CS of the schedule agent, this MEP shown in Figure B.19 is also another decision point. First of all it is a decision point from the view point of the response policy, i.e. since it is a type of event,

then it should be handled when the response policy is set to event based. But besides this, it is an important decision point in the view from the scheduling perspective. It will first activate the SCHEDULE_JD state chart with CREATE_NEWJOB event, which will create a new job in the scheduling system, i.e. the schedule agent will have the new job in its schedule now. Then it will dispatch the new job with the help of the DISPATCH-NEWJOB algorithm of the schedule agent which is shown in Figure B.20. After dispatching the new job message it handles a set of actions in the case of an event based response policy. According to the current state of the passive schedule object and agent stability parameter it either performs a set of actions described below or exits the algorithm. If the agent's stability is not set as unstable then it will perform necessary state chart activations on order to start scheduling on the passive schedule. Then according to the decision process parameter, i.e. whether it takes time, or it is instantaneous, it either registers a delta time message with RELEASE_CS event for delta time later, or it immediately releases the schedule, and dispatches it to local schedulers.

After the NEWJOB message is dispatched, it will be handled by the MEP-NEWJOB algorithm of the schedule agent. This MEP will first activate the SCHEDULE_JD state chart with the event Create_NEWJOB as in the case of SCH_JOB_AGENT, i.e. the job system agent. Then the same message will be dispatched to the SCH Renewable Resource agents.

The schedule agent will schedule the system according to the control policy. If there is a decision time introduced according to the decision process parameter of the control policy then a DELTA_TIME lag will be introduced to the decisions that will be made. To handle a decision time effect, we will have two schedules for all agents which are dealing with schedules, the active schedule, and the passive schedule. The agents' behaviour will also change according to "Agent Stability" criterion in the control policy. If the agent will be stable during the decision process, then the agent will continue to process the messages it receive with the last stable schedule on hand, while it will ignore any messages received in the unstable case.

The working of the MEP is shown in Figure B.19. A set of actions according to the control policy will be undertaken. The MEP will first check the current Job System

Response policy, whether it is set to “Event Based” or “Cyclic” response policy. If it is set to “Cyclic” then the MEP will simply end its execution, but if it is set to “Event Based” then it will check for the agent stability during decision process parameter. If this parameter will be set as unstable, i.e. if the agent will not be able to process any message during the decision process, and if the agent is currently in a decision process, i.e. in a scheduling process, then it will simply end the execution. If the agent is set to be “stable” during the decision process, then the MEP will activate first the relevant state charts with proper events in order to be able to start a decision process, i.e. to start scheduling. Finally if the current control policy for the decision process is set to “Takes Time” rather than “Instantaneous”, then the MEP will dispatch a Register_Delta_Time message, which will trigger a future RELEASE_CS event, which again will be received by the same schedule agent. With the help of this RELEASE_CS event, the schedule agent will release its prepared schedule, which we call as the passive schedule, by swapping the active and passive schedules a delta time later. If the response policy were set as “Instantaneous” then the prepared schedule will be released immediately, and the resulting schedule will be dispatched to the local scheduling agents with the help of dispatch algorithm which is also used to dispatch new job message as shown in Figure B.20.

The dispatch algorithm will send the new job message to the renewable resource agents listening channel. The flow of the NEWJOB message will be terminated finally with the MEP-NEWJOB of the SCH Renewable Resource agent, which is shown in Figure B.21. So to summarize the NEWJOB message which will be received by the simulator agent, will first let the job agent to be aware of the new job, then the scheduling agent will be aware of it, and finally the renewable resource agent will be aware of it.

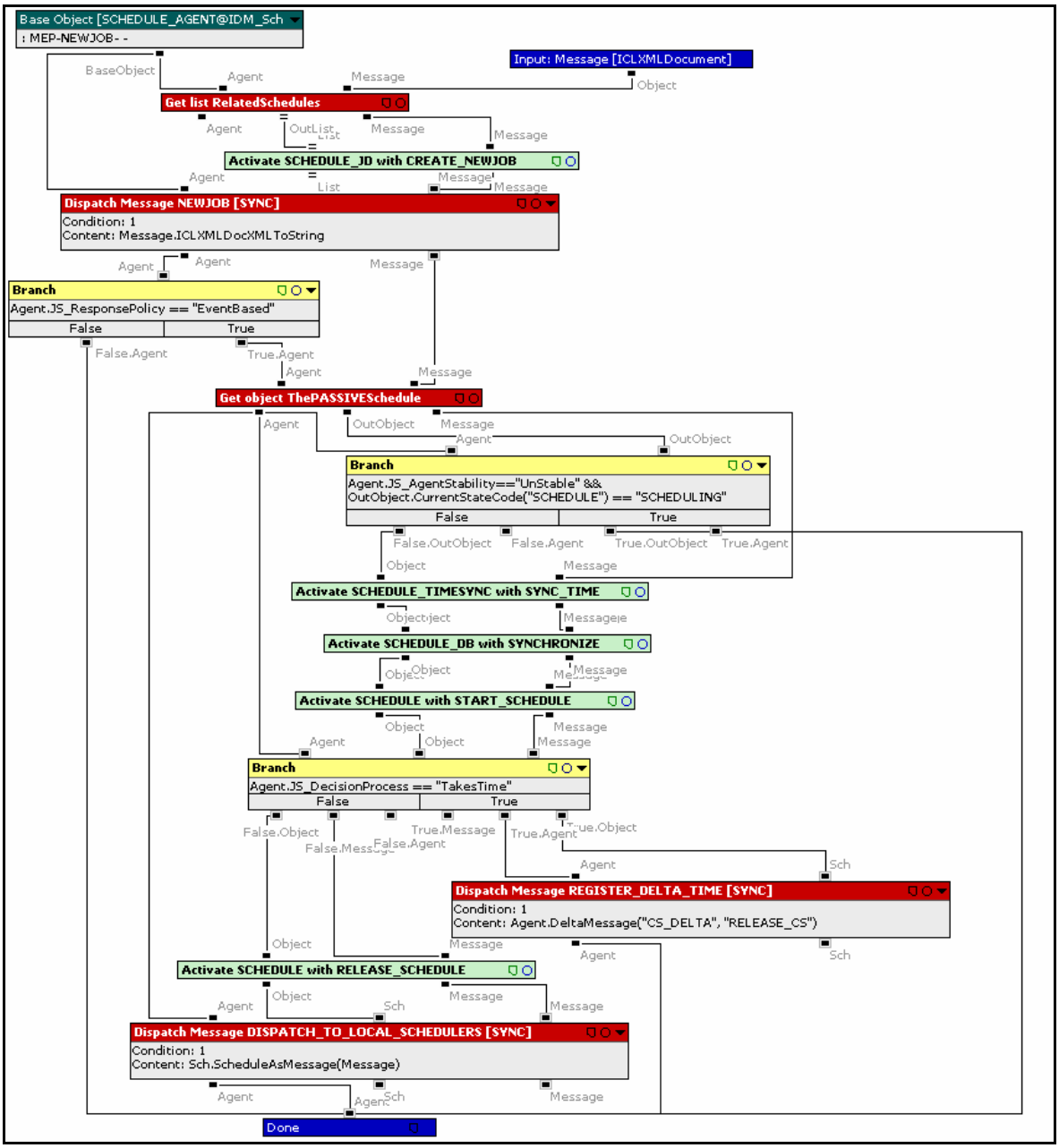

Figure B.19. MEP-NEWJOB algorithm of the Schedule_Agent

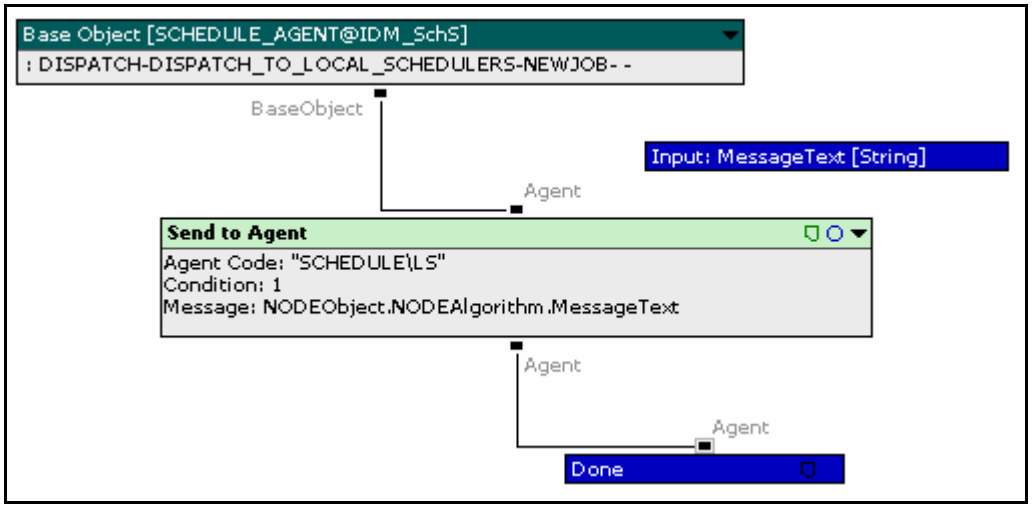

Figure B.20. DISPATCH algorithm of the Schedule Agent for NEWJOB and DISPATCH_TO_LOCAL_SCHEDULERS messages

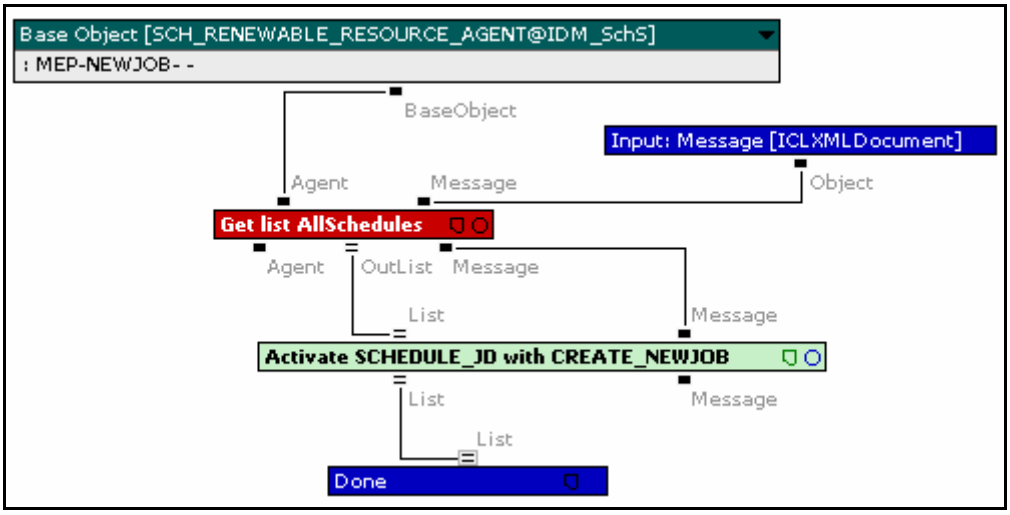

Figure B.21. MEP-NEWJOB algorithm of the SCH_RENEWABLE_RESOURCE_AGENT

B.7.3. Messaging Flow for Repair Completion Event

The messaging flow for the repair completion event can be seen in Figure B.22. The repair completion event is created by the simulator just at the instant when the breakdown event is generated. So like the breakdown event, the repair completion event is received and handled by the simulator agent. It will be first sent to the shop floor renewable resource agent, and then it is broadcasted to the scheduling agents. The scheduling

renewable resource agent will handle the repair completion event. In response to this repair completion event, it will generate an operation start event, which to be received by the shop floor renewable resource agent.

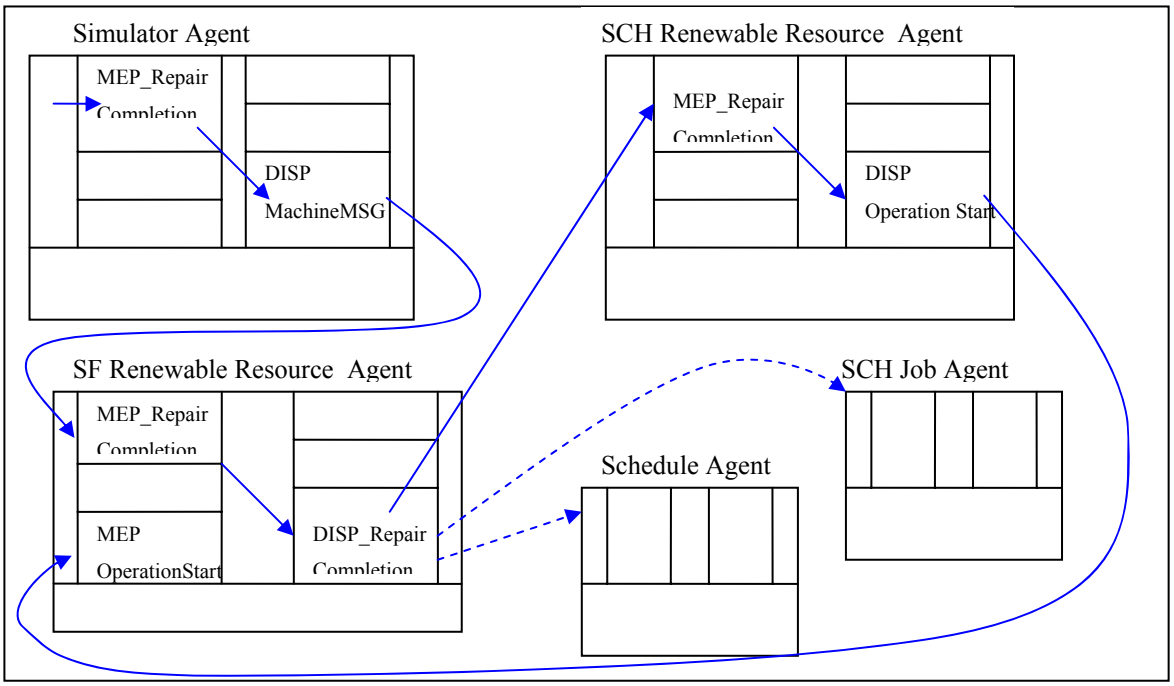

Figure B.22. Repair Completion event message flow diagram

The MEP-REPAIRCOMPLETION of the simulator agent shown in Figure B.23 will first activate the SIM_RR state chart of the SIM renewable resource object with the REPAIRCOMPLETION event. After the state chart activation, the repair completion message is dispatched with the help of DISPATCH-MACHINE_MSG algorithm, which can be seen in Figure B.12. From this point the remainder of the algorithm is the same as we discussed in the MEP-BREAKDOWN algorithm. The algorithm will check the agent response policy from the control policy parameters, and will dispatch a new CS event in the case of an event based response policy. So this MEP is another decision point implemented in this study. With the help of the dispatch machine message algorithm the message will be delivered to the SF_Renewable_Resource_Agent.

The dispatched message will be received by the MEP-REPAIRCOMPLETION algorithm of the SF_Renewable Resource Agent shown in Figure B.24. The algorithm will first find the related machine, and then will activate the SF_RR state chart with the

REPAIRCOMPLETION event. This state chart will perform an action if the object is in broken state, and if so it will at the same time create the new breakdown event. Then the SF_RR_DB state chart is activated with the UPDATE_DB event in order to synchronize the current situation to the database. Finally the REPAIRCOMPLETION message is dispatched and broadcast to the SCHEDULE channel, i.e. the schedule agent, job system agent, and renewable resource schedule agent.

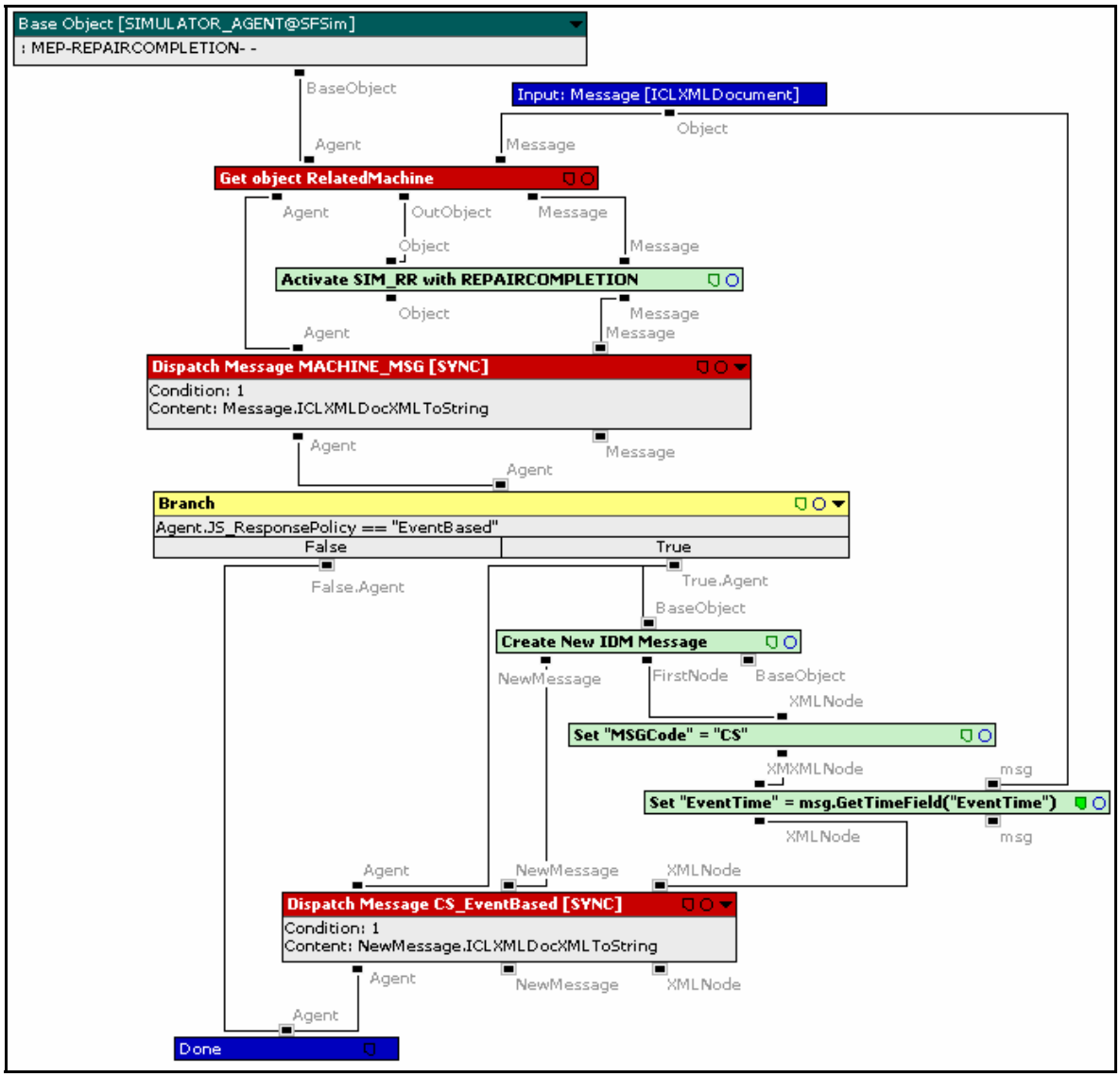

Figure B.23. MEP-REPAIRCOMPLETION algorithm of the simulator agent

The dispatched message will be handled only by the renewable resource schedule agent with the help of its corresponding MEP-REPAIRCOMPLETION algorithm, which also handles the OPERATIONCOMPLETION event, and is shown in Figure B.26. This

MEP first gets the active schedule, and activates the SCHEDULE_TIMESYNC state chart with SYNC_TIME event, i.e. it synchronizes the time of the schedule with the time of the simulator. Then the schedule will be synchronized with the database with the help of the SCHEDULE_DB state chart activation with synchronize event. After these state chart activations the related machine will be found, and an OPERATIONSTART message is dispatched.

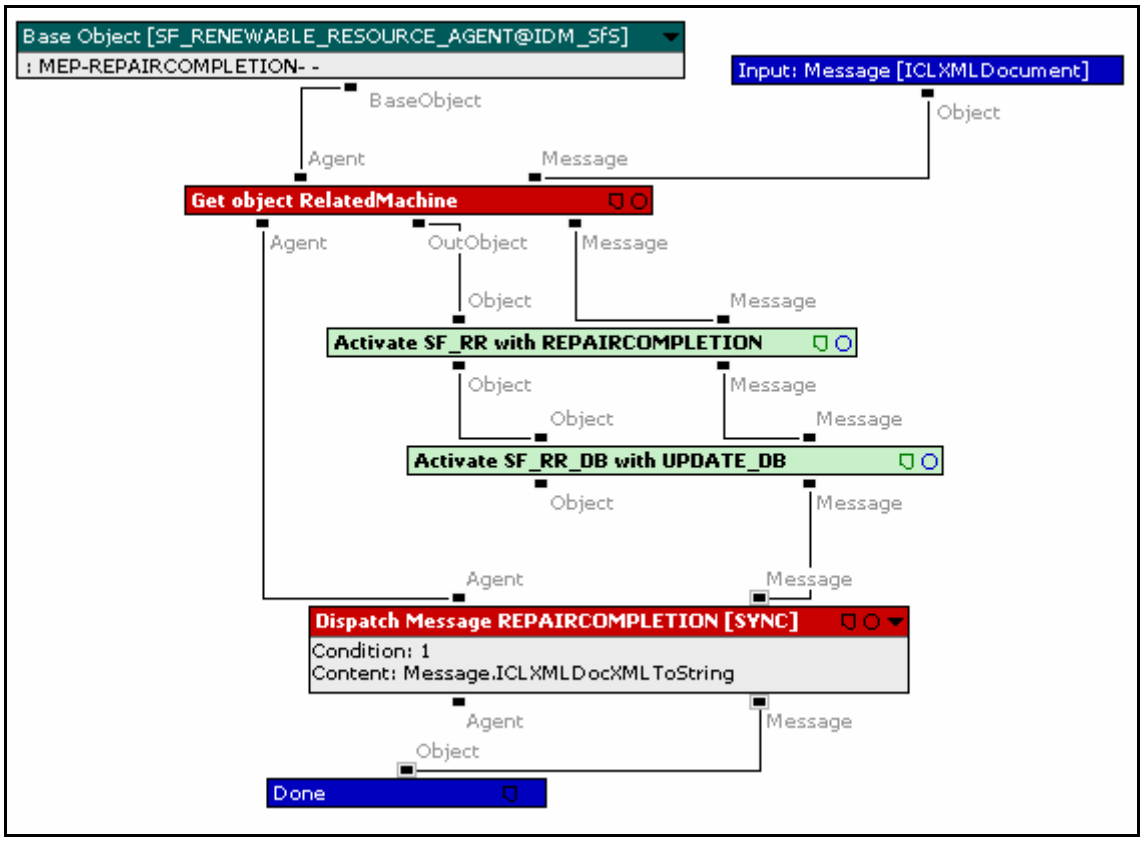

Figure B.24. MEP-REPAIRCOMPLETION algorithm of the SF_RENEWABLE_RESOURCE AGENT

The OPERATIONSTART dispatch algorithm first checks whether the related machine should start the first operation in its queue or not. If so it calls the START_OPERATION message algorithm so that the content of the message to be dispatched will be prepared. This START_OPERATION algorithm creates a new message with a message code of OPERATIONSTART and adds all the required fields to the message. Since the scheduling agent of a renewable resource group is proactive in the sense of received events, it immediately tries to adjust its situation according to the current

realizations, i.e. try to schedule the small decomposed problem. But before beginning to schedule according to current realization the agent dispatches the next available operation to be processed on the same machine according to last known good schedule, i.e. active schedule. So in this MEP one can see that the algorithm again checks agent stability condition, and the type of the decision process parameters dictated by the control policy, and perform a set of actions according to the combination of these settings.

The MEP actually takes the current passive schedule, and begins the scheduling process on it if the agent stability is not set as unstable, and the object is not in a scheduling state. If the type of the decision process is set to Instantaneous, it immediately releases the newly generated schedule by switching the active and passive schedules. After releasing the schedule it dispatches the OPERATIONSTOP and/or OPERATIONSTART messages shown in Figure B.25 by checking whether the current operation should be stopped and the first operation in the queue should be started.

This dispatch algorithm will send the OPERATIONSTART or OPERATIONSTOP message to the shop floor renewable resource agent. Then the shop floor renewable resource agent will catch this message with its relevant MEP algorithm, and the flow will be continued with this newly generated message.

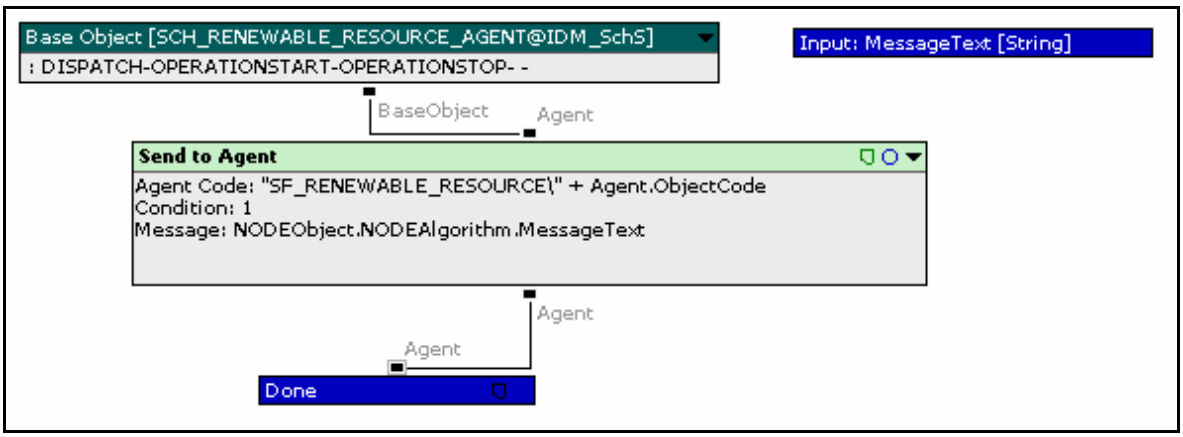

Figure B.25. DISPATCH-REPAIRCOMPLETION of the SCH_RENEWABLE_RESOURCE_AGENT

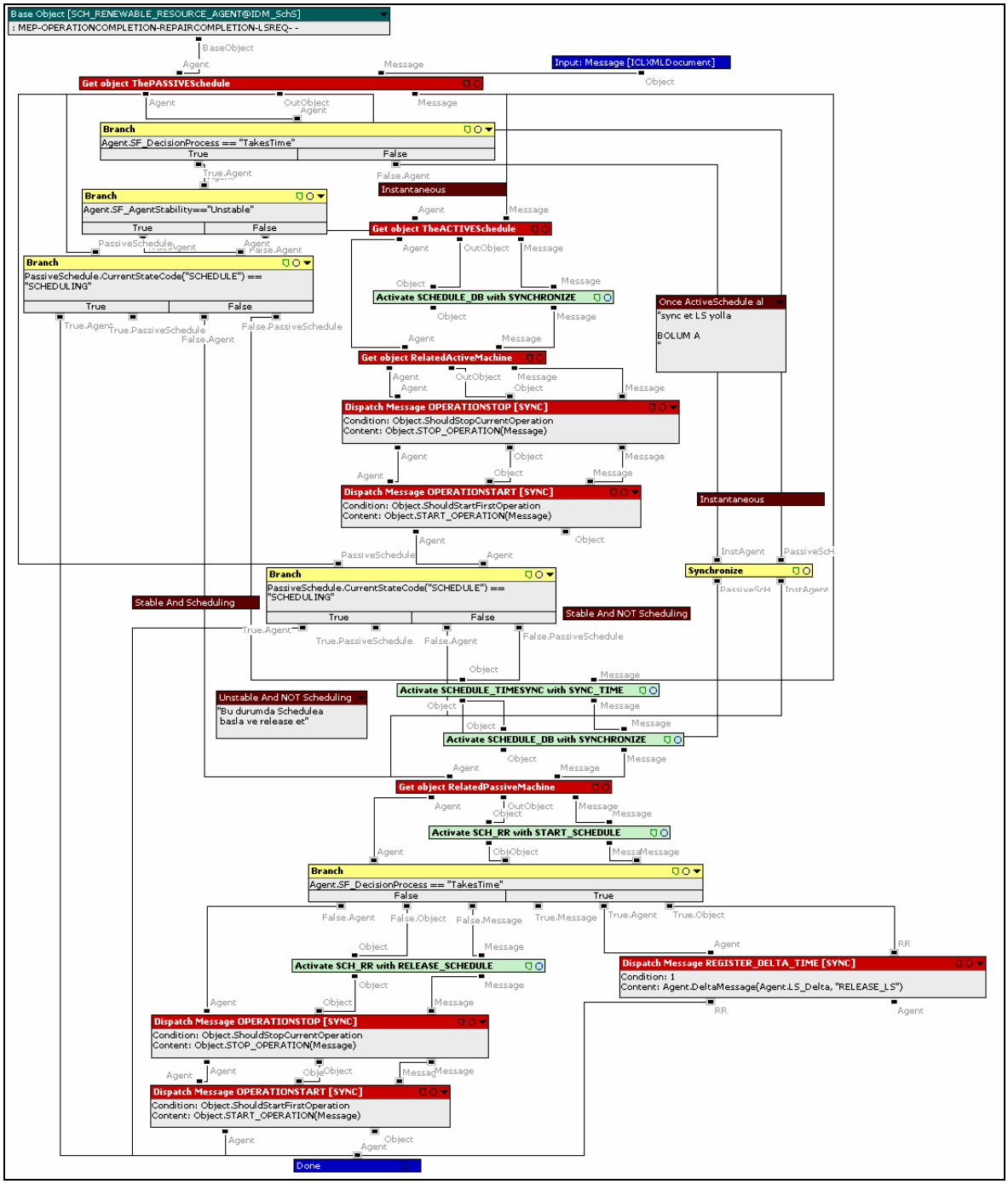

Figure B.26. MEP-OPERATIONCOMPLETION of the SCH_RENEWABLE_RESOURCE_AGENT

B.7.4. Messaging Flow for Operation Completion Event

When a shop floor renewable resource agent receives an order to start an operation, it first perform necessary actions by activating necessary state charts, and then sends an

operation start message to the simulator agent. Upon receiving this message, the simulator agent immediately activates the SIM_RR state chart with the event OPERATIONSTART. So here an estimated processing time is calculated by considering the current real operation per unit, and the batch size to be processed, and this value is added to the last start time of this operation, and the estimated finish time of the operation is found. This time will be placed into the queue as the operation completion time for this operation. Finally when the time comes, the simulator fires the operation completion event, and the simulator agent will process this message. After this initial introduction we see the messaging flow for the operation completion event in Figure B.27.

The MEP-OPEREATIONCOMPLETION of the simulator agent will process the event will dispatch it with the help of the Dispatch MachineMSG algorithm. The message will be first sent to the shop floor renewable resource agent, and then to the job system agent. The job system agent will dispatch a JOBCOMPLETION message if every operation of the related job is finished, and continue to dispatch the operation completion message to the scheduling agent of the renewable resource. Here according to current control policy and to the active and passive schedules it will dispatch operation stop and/or operation start messages to the shop floor renewable resource agent.

The MEP-OPERATIONCOMPLETION of the simulator agent shown in Figure B.28 will first activate the SIM_RR state chart of the SIM renewable resource object with the OPERATIONCOMPLETION event. After the state chart activation, the operation completion message is dispatched with the help of DISPATCH-MACHINE_MSG algorithm, which can be seen in Figure B.12. From this point the remainder of the algorithm is the same as we discussed in the MEP-BREAKDOWN and MEP-REPAIRCOMPLETION algorithms. The algorithm will again check the agent response policy from the control policy parameters, and will dispatch a new CS event in the case of an event based response policy. So this MEP is another decision point implemented in this study. With the help of the dispatch machine message algorithm the message will be delivered to the SF_Renewable_Resource_Agent.

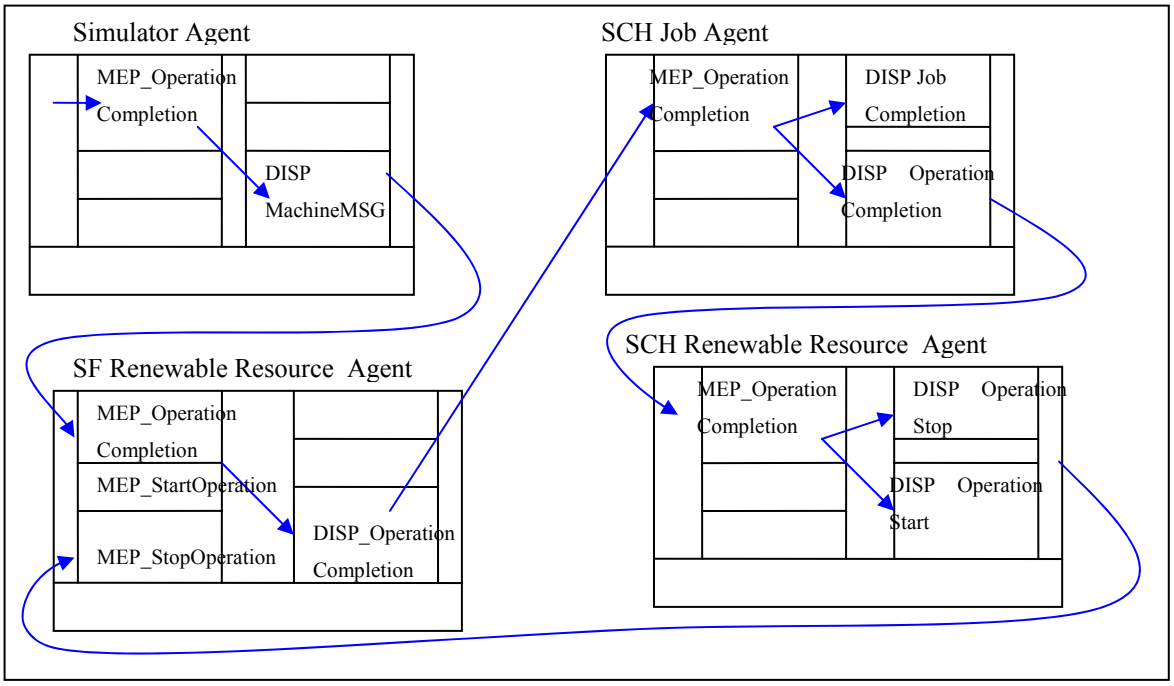

Figure B.27. Operation Completion event message flow diagram

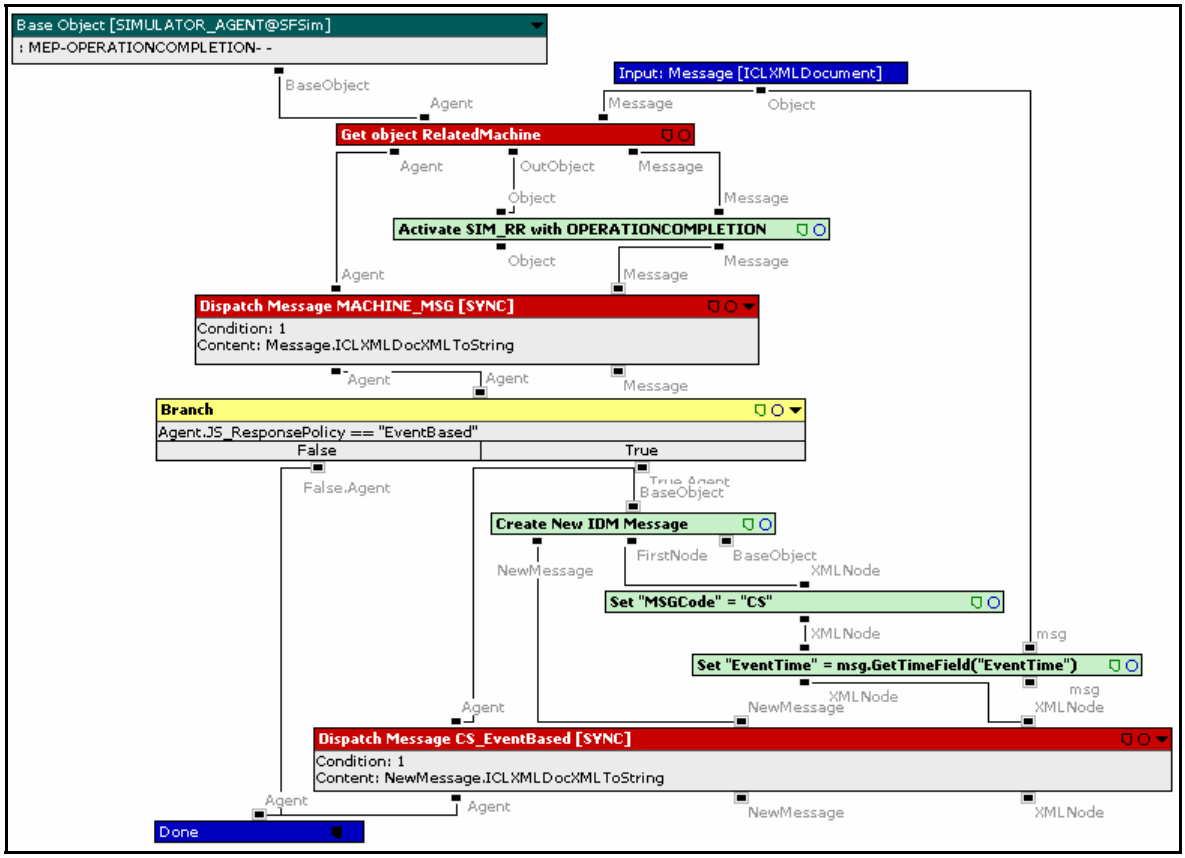

Figure B.28. MEP-OPERATIONCOMPLETION algorithm of the Simulator Agent

The dispatched message will be received by the MEP-REPAIRCOMPLETION algorithm of the SF_Renewable Resource Agent shown in Figure B.29. The algorithm will first find the indicated machine, and its current operation. If it has an operation on it, it changes the state of the SF_Operation object to CLOSED state by activating the SF_OP state chart with the OPERATIONCOMPLETION event. Then the SF_OP_DB state chart is activated to update the database, and finally the SF_RR state chart of the shop floor renewable resource object is activated with OPERATIONSTOP event so that the state of this object make a transition to the idle state. Upon the execution of these state charts the OPERATIONCOMPLETION message is dispatched by the dispatch algorithm shown in Figure B.30 to the job system agent with the content generated by the OPERATIONCOMPLETION message algorithm.

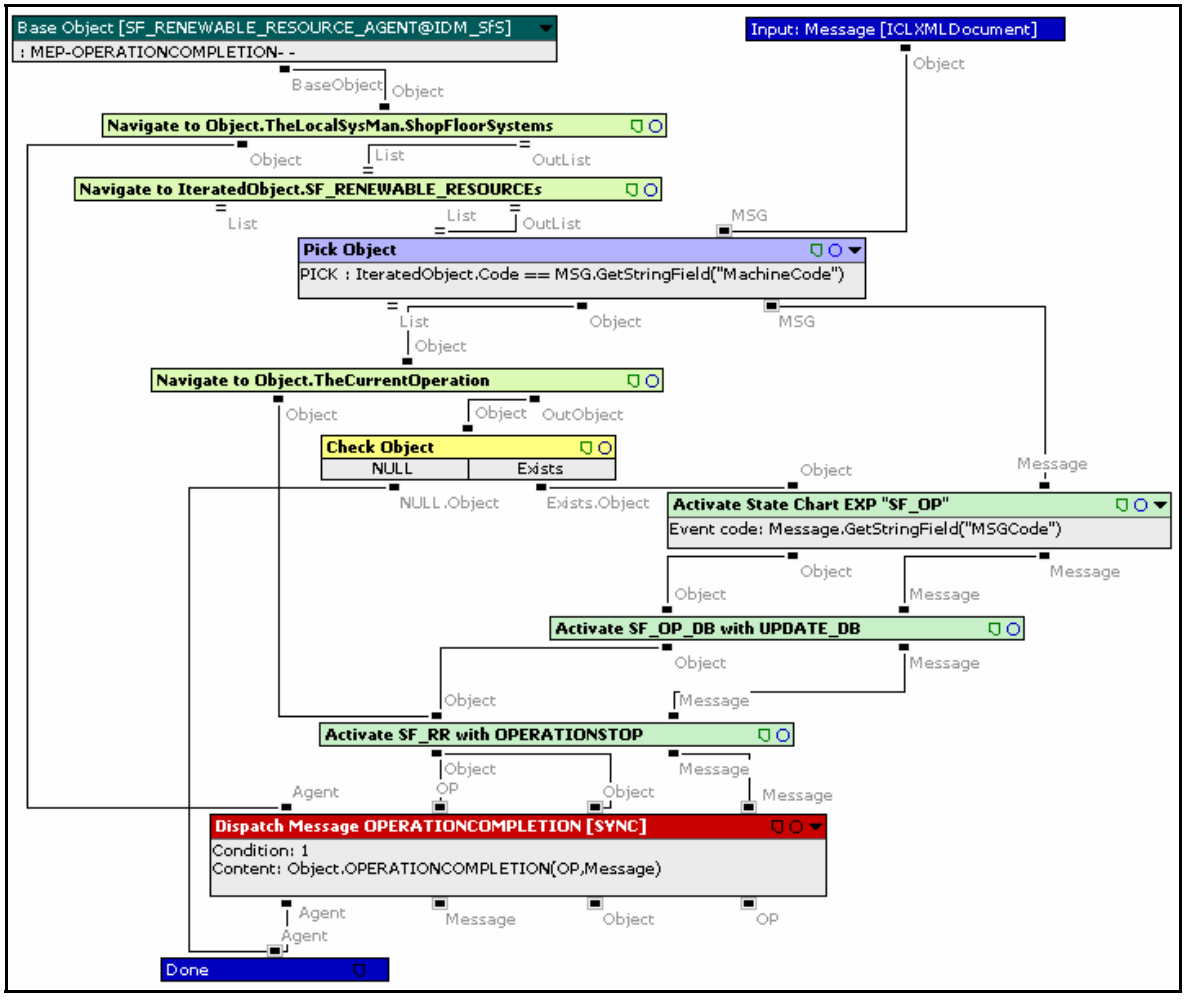

Figure B.29. MEP-OPERATIONCOMPLETION algorithm of the SF_RENEWABLE_RESOURCE AGENT

The MEP-OPERATION completion algorithm of the job system agent is shown in Figure B.31. The algorithm will first synchronize the schedule with the database, and after finding the related job it will change the state of the SCH_JOB aspect of the job system agent to COMPLETED state. Then it further checks whether the all the operations of the related job is finished, and if so dispatches a JOBCOMPLETION message with. Finally it dispatches the OPERATIONCOMPLETION message with the help of the dispatch algorithm shown in Figure B.32 to the SCH_Renewable Resource Agent.

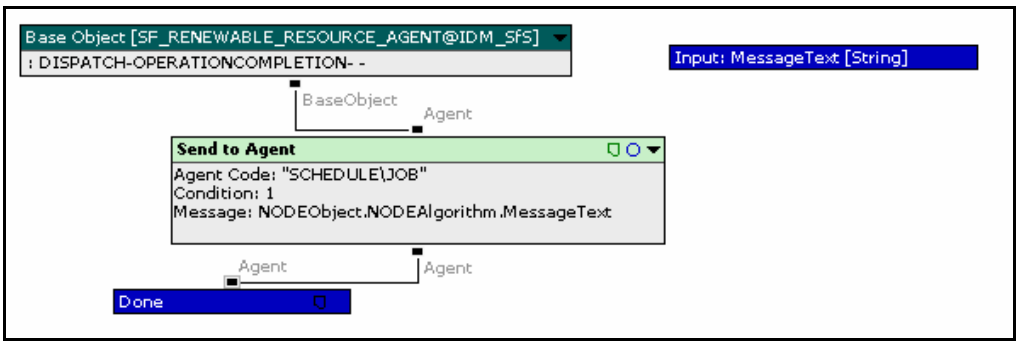

Figure B.30. DISPATCH-OPERATIONCOMPLETION algorithm of the SF_RENEWABLE RESOURCE_AGENT

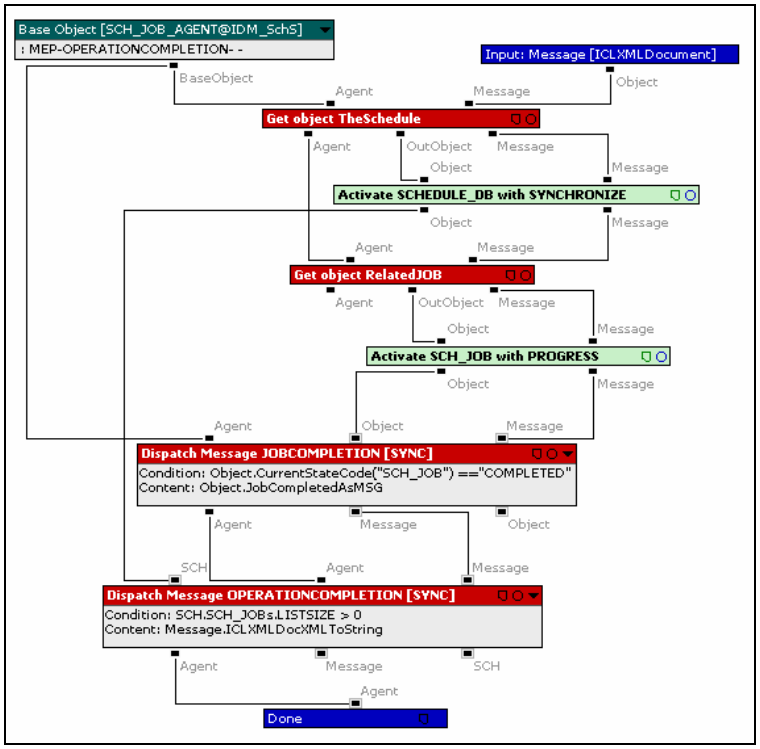

Figure B.31. MEP-OPERATIONCOMPLETION of the SCH_JOB_AGENT

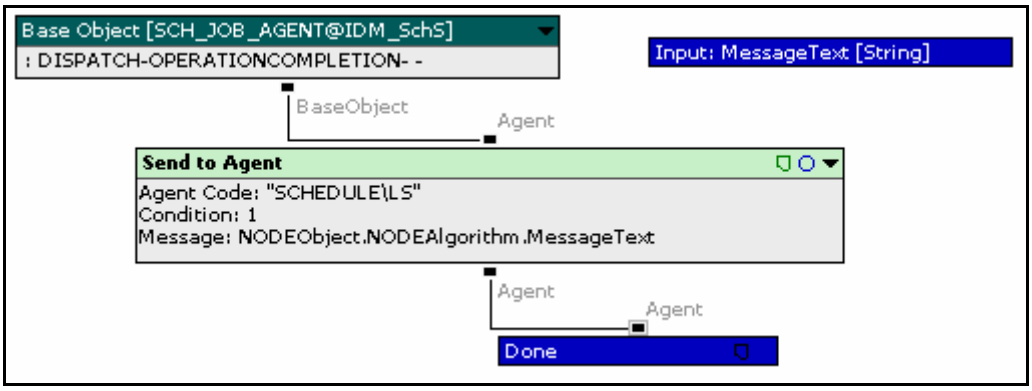

Figure B.32. DISPATCH-OPERATIONCOMPLETION algorithm of the SCH_JOB_AGENT

The MEP-OPERATIONCOMPLETION of the renewable resource scheduling agent, which also handles the repair completion event shown in Figure B.26, receives this message. The working of this algorithm is discussed in section B.7.3, by the explanation of Figure B.26 in the repair completion message flow section. To summarize this algorithm starts the first operation according to the active schedule currently available, and then according to the criterions defined by the control policy it either registers a delta time for a RELEASE_CS event, or simply dispatches the stop message for the current operation and/or dispatches a start message for the first available operation.

When the JOBCOMPLETION message is dispatched shown in Figure B.31 with the message code as JOBCOMPLETED, this message is sent to the simulator agent. This is a special message for the experimental setup since all the statistics related to the experiment will be collected when a SIMULATIONRUNCOMPLETED message is dispatched. The message content is generated by the ReplicationCompletesAsMSG algorithm with a message code of SIMULATIONCOMPLETED in response to this message by the MEP-JOBCOMPLETED algorithm of the simulator agent shown in Figure B.33. This message is dispatched with the dispatch algorithm shown in Figure B.34, and sent to all agents. Sending the message to all agents is realized since the value of the receiver agent in the Send To Agent node is intentionally left blank.

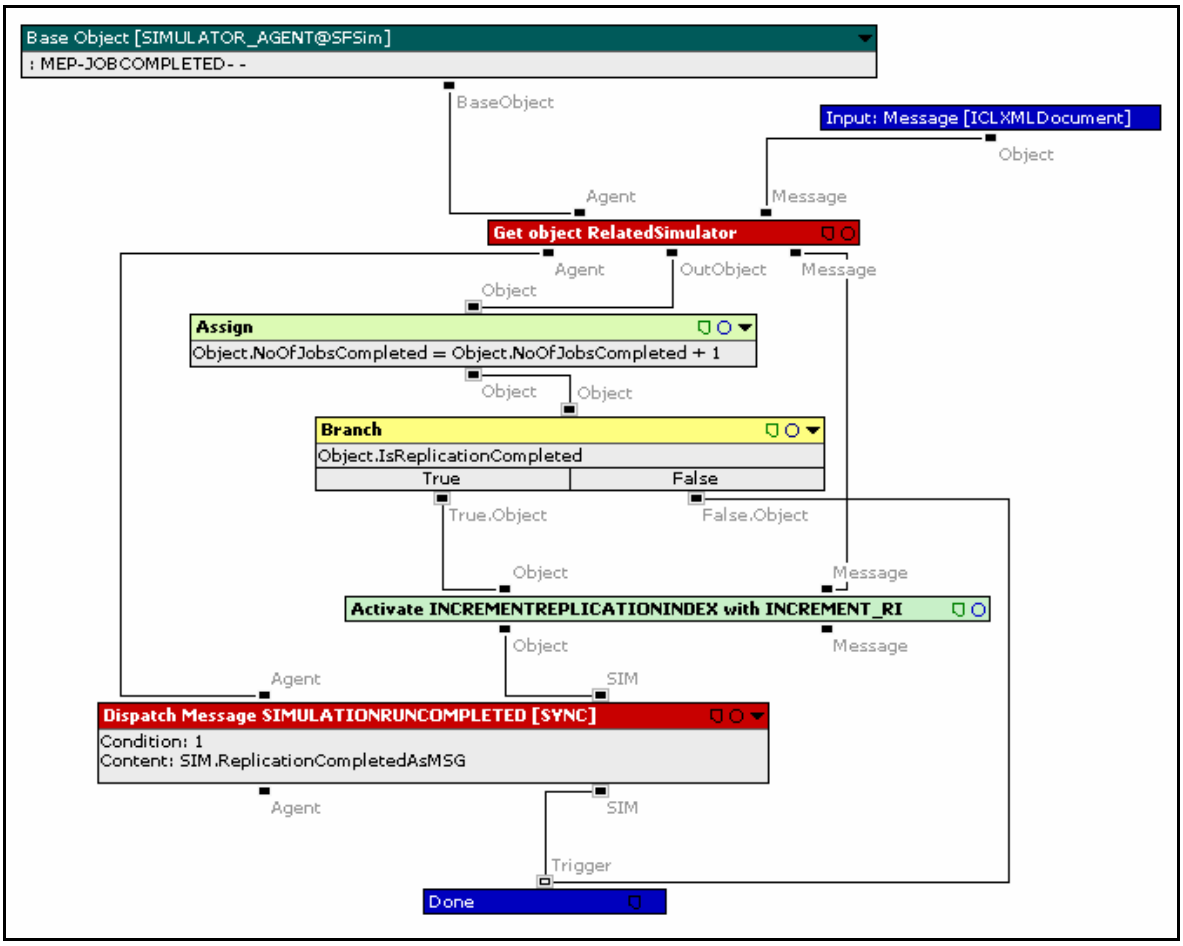

Figure B.33. MEP-JOBCOMPLETED of the Simulator Agent

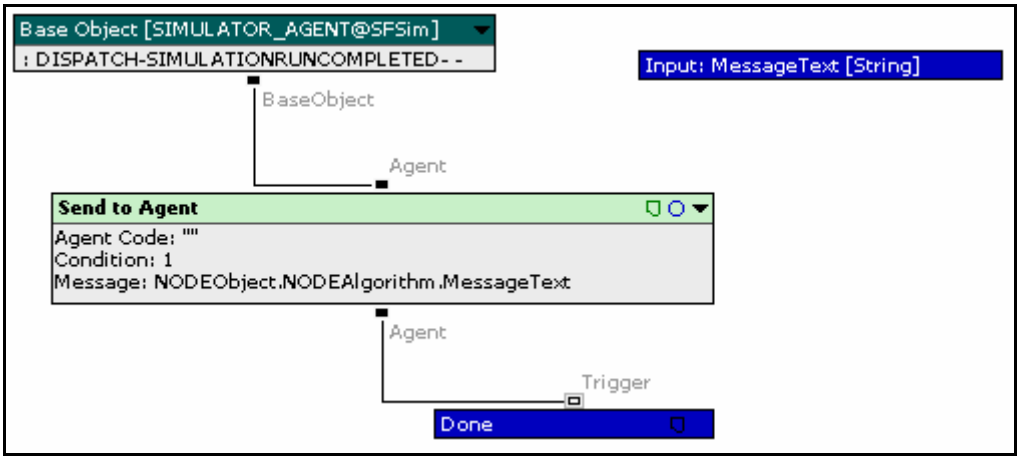

Figure B.34. DISPATCH-SIMULATIONRUNCOMPLETED of the Simulator Agent

When this message is received by the all three scheduling agents, i.e. the schedule agent, job system agent and the renewable resource schedule agent, they will do memory

cleanup operations in order to prepare themselves for the next simulation steps. In addition to these actions, the job system agent collects the required statistics to be evaluated as experimental results.

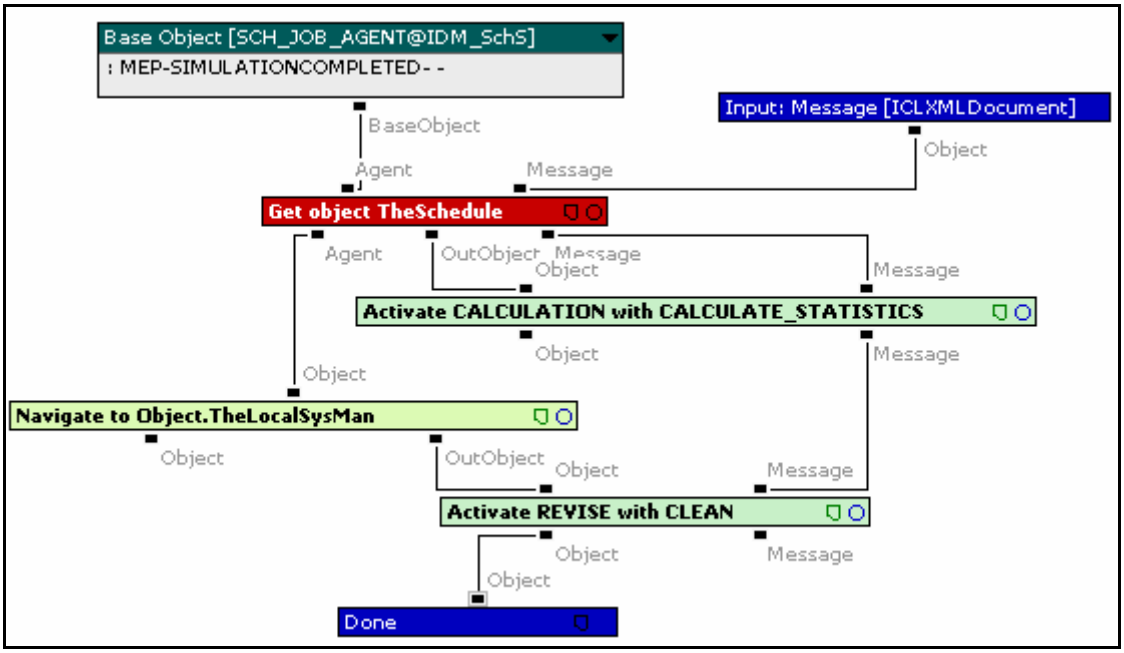

Figure B.35. MEP-SIMULATIONCOMPLETED algorithm of the SCH_JOB_AGENT

B.8. Messaging in ICRON

The messaging mechanism established in the ICRON model of the proposed scheduling system handles the events occurring in the shop floor by building messaging channels between system entities. By using these messaging channels, the system entities communicate with each other and give or take information about their current states, and by using this information, they reschedule their operations. Since this messaging mechanism is achieved in a dynamic and distributed manner, the changes and disturbances in the shop floor can be responded immediately.

The messages are formed as XML documents. Thanks to the flexible structure of XML one can easily define tree like data structures, and easily form messages containing any kind of data like large schedule data. Although the documents are formed in XML format, they are first converted to hashed strings, and then transmitted via the channels in

the messaging system. The received agent will then re-transform the received message string to its original XML document, and so it can easily manipulate the received XML message with the help of its built-in message functions.

In the implementation a message code field is added to every created message document for the sake of a generic implementation. In addition to that also an event time field is added. So almost every message has a “Message Code” and “Event Time” keys in common. With the help of the timestamp field the sent and received messages during the runs of these simulations will be analyzed.

There are two types of messages. The first one can be classified as events, and are generated by the simulator, and the second one consists of standard messages which are generated as a result of some actions.

B.8.1. BREAKDOWN Message Declaration

The breakdown message is an event type of message, and is generated by the simulator. This message is generated in the Handle algorithm of the BREAKDOWN event discussed in section B.6.1.1. The breakdown message includes the following key value pairs;

- MSGCode : BREAKDOWN
- EventTime: The time of this event when it will be realized.
- MachineCode : The code of the renewable resource which will break down.
- EstimatedBreakDownDuration : As the key implies, it holds the estimated duration of this breakdown.

B.8.2. NEWJOB Message Declaration

The NEWJOB message is an event type of message, and is generated by the simulator. This message is generated in the Handle algorithm of the NEWJOB event

discussed in section B.6.1.2. Before going into the details of the message, it would be appropriate to explain the structure of a job in the system.

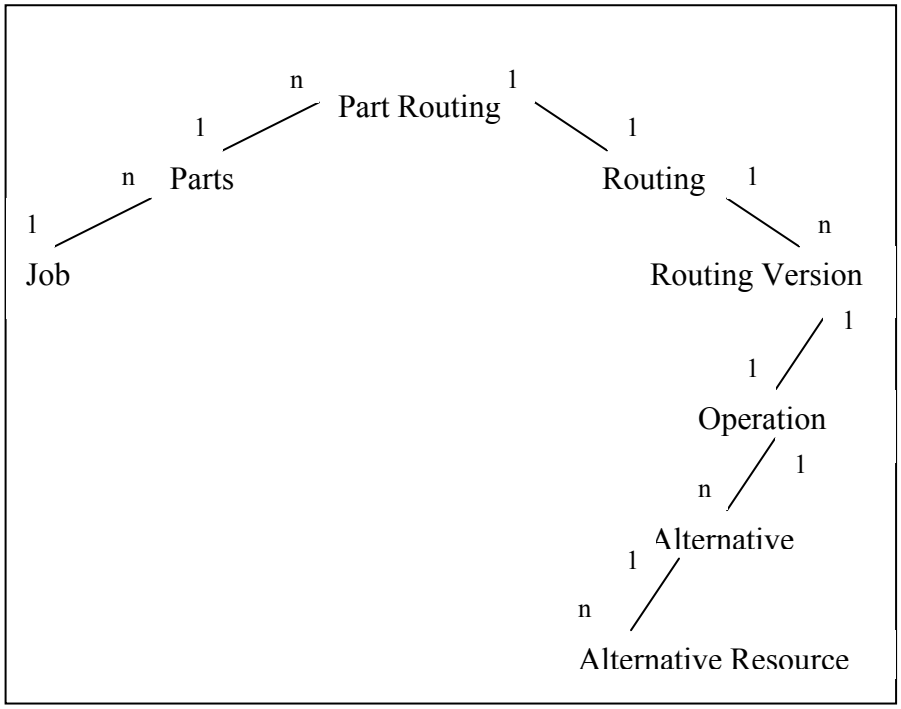

Figure B.36. Representation of the relationships of a new job and its building blocks

A part is defined as a semi-finished or finished product that can be stored in the inventory. Orders and jobs in a scheduling system are defined for parts. Every part to be produced has to be assigned a routing that will describe its production phases.

The Part Routing defines all the routes with which a part can be produced, so a part may have many routings. The Routing defines the set and sequence of operations that a part should undergo to become a finished or a semi- finished product. As an example, in order to have tempered glass, first one should cut the glass plate to the shape, then temper it, drill holes on it and finally inspect it. Usually a routing is composed of multiple operations. There can be operations running in parallel or in series. In such a case, operations have to be linked to each other by predecessor and successor relations. Unless it is not the first operation of a routing, every operation has to be linked to its predecessor that should finish completely or partly before its start. Similarly, unless it is not the last operation of a routing, every operation has to be linked to its successor that can start after

its complete or partial finish. A Routing may have many versions and they are defined in the Routing Version.

An operation is the elementary constituent of a part's routing. It defines all production related details required to perform a particular manufacturing step in the routing. For example drilling a hole is an operation of the glass tempering routing. Operations run either on units or batches of parts.

Each of the alternative ways of accomplishing an operation is called a processing alternative, and an operation may have many alternatives. Until scheduling is performed it is not possible to state on which processing alternative the operation will take place. It is indeed the goal of scheduling to make the decision on resource selection based on processing alternative definitions. Operation details such as resource requirements for renewable resources and consumables, setup requirements, operation times, waiting times, scrap rates, transfer batch size are all defined separately for every processing alternative.

An operation of a part can be accomplished in various ways. This fact may be due to parallel identical or substitute resources present in the manufacturing environment. For example one can drill a hole either with a manual machine or an automatic machine. So an alternative process may be processed by many alternative resources, so an operation can be processed on many alternative resources.

With help of this clarifications, now a new job message can be defined with the following key value pairs;

- MSGCode : NEWJOB
- EventTime: The time of this event when it will be realized.
- JobSystem: It holds the job system code. In this study for the sake of simplicity we use a default value of IDM_JS.
- JobCode: It represents the job code, and we simply create a string which is generated by the concatenation of the string "JOB-" and a GUID (Globally Unique Identifier), and this code will be unique no matter how many new jobs will arrive to the system.

- OrderCode: It represents the order code. In this study for the sake of simplicity we use a default value of ORDER1.
- ProductSystemCode: It represents the product system code.
- PartCode: It represents the part code.
- RoutingSystemCode: It represents the routing system code.
- RoutingCode : It represents the routing code.
- RoutingVersion: It represents the routing version.
- nBatch: It represents the batch size of the job. It is generated according to the minimum and maximum batch size values, and a given distribution.
- DueDate : It stands for the due date of the job. This is also a generated time according to the problem parameters. There is a due date scaling factor, with which we can control this value, according to the due date tightness problem parameter.
- ReleaseTime: It represents the release time of the job.

B.8.3. REPAIRCOMPLETION Message Declaration

The repair completion message is an event type of message, and is generated by the simulator. This message is generated in the Handle algorithm of the REPAIRCOMPLETION event discussed in section B.6.1.3. The repair completion message includes the following key value pairs;

- MSGCode : REPAIRCOMPLETION
- EventTime: The time of this event when it will be realized.
- MachineCode : The code of the renewable resource which will be repaired.

B.8.4. OPERATIONCOMPLETION Message Declaration of the Simulator Agent

The operation completion message is an event type of message, and is generated by the simulator to trigger the operation completion MEP's of relevant agents. This message is generated in the Handle algorithm of the OPERATIONCOMPLETION event discussed in section B.6.1.4. The operation completion message includes the following key value pairs;

- MSGCode : OPERATIONCOMPLETION
- EventTime: The time of this event when it will be realized.
- MachineCode : The code of the renewable resource, i.e. machine, on which the current operation is completed. It may seem a little bit confusing to send the machine code instead of the operation code, but it is implemented in that way in our study.

B.8.5. REGISTER_DELTA_TIME Message Declaration

The REGISTER_DELTA_TIME message is a special message, which is used to register a delta event to the system, which will be fired at a future time with given message code, and receiver agent. It is created with the help of the DeltaMessage algorithm shown in Figure B.8. It can be thought as the trigger for setup of an alarm clock, and with this message an alarm clock is set up to signal an alarm at a future time. The only difference is, since one supplies the callback function with the supplied message code, and receiver agent values, it will perform an action at this predetermined future time. The register delta time message includes the following key value pairs;

- MSGCode : REGISTER_DELTA_TIME
- EventTime: It represents the current time. It will not be used in the system for any purpose, it is there for tracking purposes only.
- ReturnMSG: It represents the message code of the message that should be released at a future time.
- Receiver Agent: It represents the receiver agent.
- DeltaCode : It represents the type of the delta event. In this case it can be CS_CYCLE, CS_DELTA or LS_DELTA. In the system database in the DELTATIMES table, these keys have time values. According to the time values specified in the database, they are used to determine the future time of the specified delta event. CS_CYCLE code is used to enable the cyclic response policy implementation of the specified control policy. CS_DELTA and LS_DELTA are used to enable the decision time effect for their relevant scheduling agents.

B.8.6. DELTA_MSG Message Declaration

The Delta message is an event type of message, and is generated by the simulator when a REGISTER_DELTA_TIME message is dispatched to the simulator, and it is handled by the MEP-REGISTER_DELTA_TIME algorithm of the simulator agent. This message is generated in the Handle algorithm of the DELTA event discussed in section B.6.1.5. The operation completion message includes the following key value pairs;

- MSGCode : DELTA_MSG
- EventTime: The time of this event when it will be realized.
- ReturnMSG : It represents actually the message code of the message that should be released at a predetermined time, i.e. at the time defined by the EventTime.
- ReturnPath : It represents the receiver agent.

So to summarize, to be able to perform an action in the future we use the DELTA event mechanism. First a REGISTER_DELTA_TIME message is dispatched to the simulator agent. By receiving this message the simulator agent creates a new delta event with current time, and with the ReturnMSG and ReturnPath values supplied by the REGISTER_DELTA_TIME messages ReturnMSG and ReceiverAgent values. So since this event is placed into the simulators queue with current time, it will be first processed with the HANDLE algorithm of the Delta event discussed in B.6.1.5. With this algorithm the above mentioned message will be released.

B.8.7. CS Message Declaration

The CS message triggers the response of the schedule agent. In the case of an event based response policy the system fires a CS message on every event, so that the job system will be responsive. The CS message includes the following key value pairs;

- MSGCode : CS
- EventTime: The time of this event when it will be realized.

B.8.8. OPERATIONSTART Message Declaration

The Operation Start message is created with the START_OPERATION algorithm of the SCH Renewable Resource, and this algorithm is a special SC_MSG algorithm, i.e. algorithm with the context SC_MSG discussed in A.4.3. The operation start message includes the following key value pairs;

- MSGCode : OPERATIONSTART
- EventTime: It represents the current time of the simulator.
- MachineCode: It represents the code of the renewable resource.
- JobSystem: It holds the job system code.
- JobCode: It represents the job code.
- OPTemplateCode: It represents the operation template code.
- SplitIndex: It represents the split index of the operation. Under normal conditions, operation batches are processed as a whole. However, in some cases operation batches may be divided into several pieces for being processed in different times or in parallel on identical resources.
- OperationTimePerUnit: As the name implies, it represents the operation time per unit.
- BatchSize: It represents the current batch size to be scheduled.
- AlternativeCode: It represents the code of the alternative code for alternative processing of the operation.

B.8.9. OPERATIONSTOP Message Declaration

The Operation Stop message is created with the STOP_OPERATION algorithm of the SCH Renewable Resource, and this algorithm is also a special SC_MSG algorithm, i.e. algorithm with the context SC_MSG discussed in A.4.3. The operation stop message includes the following key value pairs;

- MSGCode : OPERATIONSTOP
- EventTime: It represents the current time of the simulator.

- MachineCode: It represents the code of the renewable resource.
- JobSystem: It holds the job system code.
- JobCode: It represents the job code.
- OPTemplateCode: It represents the operation template code.
- SplitIndex: It represents the split index of the operation.

B.8.10. OPERATIONCOMPLETION Message Declaration of the Shop Floor Renewable Resource Agent

This operation completion message is generated by the SC_MSG type OPERATIONCOMPLETION algorithm of the shop floor renewable resource agent. The operation completion message includes the following key value pairs;

- MSGCode : OPERATIONCOMPLETION
- EventTime: It represents the event time.
- MachineCode: It represents the code of the renewable resource.
- JobSystem: It holds the job system code.
- JobCode: It represents the job code.
- OPTemplateCode: It represents the operation template code.
- SplitIndex: It represents the split index of the operation.

B.8.11. JOBCOMPLETED Message Declaration

The JOBCOMPLETED message is used as a trigger message like the CS message. As discussed in section B.7.4, the job completed message is used to trigger the problem indexes of the experiment, and if necessary to trigger the calculation of the experiment statistics. The JOBCOMPLETED message includes the following key value pairs;

- MSGCode : JOBCOMPLETED
- EventTime: It represents the current time of the simulator.

B.8.12. CENTRAL_SCHEDULE Message Declaration

It is one of the most important message in this study. It holds all the operations of the current schedule generated by the scheduling agent. The central schedule message is created with the ScheduleAsMessage algorithm of the Schedule object, and this algorithm is also a special SC_MSG algorithm. It contains the schedule which will be dispatched to the local schedulers. The central schedule message includes the following key value pairs;

- MSGCode : CENTRAL_SCHEDULE
- EventTime: It represents the current time of the simulator.
- Operations List: All the scheduled operations will be added to this operations list of the message. Below are the fields related to each operation.
  - MachineCode: It represents the code of the renewable resource.
  - JobSystem: It holds the job system code.
  - JobCode: It represents the job code.
  - OPTemplateCode: It represents the operation template code.
  - SplitIndex: It represents the split index of the operation.
  - RunST: It represents the scheduled start time of the operation.
  - RunCT: It represents the scheduled completion time of the operation.

In this message the flexibility of the XML document is enjoyed. One can simply send many operations attached to one message code with the help of this XML structure.

B.8.13. SIMULATIONCOMPLETED Message Declaration

The SIMULATIONCOMPLETED message is used as a trigger message like the CS and JOBCOMPLETED messages. As discussed in section B.7.4, the simulation completed message is used to trigger the calculation of the experiment statistics. The SIMULATIONCOMPLETED message includes the following key value pairs;

- MSGCode : SIMULATIONCOMPLETED
- EventTime: It represents the current time of the simulator.
- ProblemSettingID: It represents the current problem setting ID.

- ReplicationIndex: It represents the current replication index of the simulated problem setting ID.

REFERENCES

Aderounmu, G.A., 2004, "Performance comparison of remote procedure calling and mobile agent approach to control and data transfer in distributed computing environment", *Journal of Network and Computer Applications*, Vol 27. pp. 113-129.

Attanasio, A., G., Ghiani, L., Grandinetti, F., Guerriero, 2006, "Auction algorithms for decentralized parallel machine scheduling", *Parallel Computing*, Vol 32, pp 701-709.

Baker, K.R., and J.W.M., Bertrand, 1981, "An Investigation Of Due-Date Assignment Rules With Constrained Tightness", Journal of Operations Management, Vol 1, No 3, pp. 110-120.

Baker, K.R., and J.W.M., Bertrand, 1982, "A Dynamic Priority Rule for Scheduling Against Due-Dates", *Journal of Operations Management*, Vol 3, No 1, pp. 37-42.

Baker, K.R., and J.J., Kanet, 1983, "Job Shop Scheduling With Modified Due Dates", *Journal of Operations Management*, Vol 4, No 1, pp. 11-22.

Baker, K.R., 1984, "Sequencing rules and due-date assignments in a job shop", *Management Science*, Vol 30, No 9, pp. 1093-1104.

Baker, A. D., 1998, "A survey of factory control algorithms that can be implemented in a multi-agent heterarchy: dispatching, scheduling, and pull", *Journal of Manufacturing Systems*, Vol 17, No 4, pp 297 – 320.

Barroso, A.M., J.C.B., Leite, and O.G., Loques, 2002, "Treating uncertainty in distributed scheduling", *The Journal of Systems and Software*, Vol 63. pp. 129-136.

Brennan, R.W., and D.H., Norrie, 2001, "Evaluating the performance of reactive control architectures for manufacturing production control", *Computers in Industry*, Vol 46, pp 235- 245.

Brennan, R.W, and D.H., Norrie, 2003, "Metrics for evaluating distributed manufacturing control systems", *Computers in Industry*, Vol 51. pp. 225-235.

Campos, A.E., and , J.E, Navarro., 2004, "A page-coherent, casually consistent protocol for distributed shared memory", *The Journal of Systems and Software*, Vol 72. pp. 305-319.

Cao, J., X., Wang, and S. K., Das, 2004, "A framework of using cooperating mobile agents to achieve load sharing in distributed web server groups", *Future Generation Computer Systems*, Vol 20. pp. 591-603.

Cavalieri, S., M., Garetti, M., Macchi, and M., Taisch, 2000, "An experimental benchmarking of two multi-agent architectures for production scheduling and control", *Computers in Industry*, Vol 43, pp 139- 152.

Cesta, A., and D., D'aloisi, 1999, "Mixed-Initiative Issues in an Agent-Based Meeting Scheduler", *User Modeling and User-Adapted Interaction*, Vol 9, pp. 45-78.

Chun, A., H., Wai, and R.Y.M., Wong, 2003a, "Optimizing agent-based meeting scheduling through preference estimation", *Engineering Applications of Artificial Intelligence*, Vol 16. pp. 727-743.

Chun, H.W., and R.Y.M., Wong, 2003b, "N*—an agent-based negotiation algorithm for dynamic scheduling and rescheduling", *Advanced Engineering Informatics*, Vol 17, pp. 1-22.

Chun, A., H., Wai, Y.M., Rebecca, and Y.M., Wong, 2003, "Optimizing agent-based meeting scheduling through preference estimation", *Engineering applications of Artificial Intelligence*, Vol 16, pp. 727-743.

Church, L.K, and R., Uzsoy, 1992, "Analysis of periodic and event-driven rescheduling policies in dynamic shops", *International Journal of Computer Integrated Manufacturing*, Vol 5, No 3, pp. 153-163.

Garey ,M. R., and D. S., Johnson, 1979, "*Computers and Intractability: A Guide to the Theory of NP-Completeness*" New York, NY: W. H. Freeman and Company.

Fabre, J.C., and T., Perennou, 1997, "Processing of confidential information in distributed systems by fragmentation", *Computer Communications*, Vol 20. pp. 177-188.

Ferber, J., 1999, *Multi-agent systems: An introduction to Distributed Artificial Intelligence*, Addison-Wesley, London.

Harchol-Barter, M., M.E., Crovella, and C.D., Murta, 1999, "On Choosing a Task Assignment Policy for a Distributed Server System", *Journal of Parallel and Distribued Computing*, Vol 59. pp. 204-228.

Jennings, N.R., P., Faratin, A.R., Lomuscio, S., Parsons, M., Wooldridge, and C., Sierra, 2001, "Automated Negotiation: Prospects, Methods and Challenges", *Group Decision and Negotiation*, Vol 10, pp 199- 215.

Jeong, I., and V.J., Leon, 2005, "A single-machine distributed scheduling methodology using cooperative interaction via coupling agents", *IIE Transactions*, Vol 37, pp. 137-152.

Karageorgos, A., N., Mehandjiev, G., Weichhart, and A., Hammerle, 2003, "Agent-based optimisation of logistics and production planning", *Engineering Applications of Artificial Intelligence*, Vol 16. pp. 335-348.

Kim, K., and B.C., Paulson, 2003, "Agent-Based Compensatory Negotiation Methodology to Facilitate Distributed Coordination of Project Schedule Changes", *Journal of Computing in Civil Engineering*, Vol 17, No 1, pp. 10-18.

Kutanoglu, E., and I., Sabuncuoglu, 2001, "Experimental Investigation of Iterative Simulation-Based Scheduling in a Dynamic and Stochastic Job Shop", *Journal of Manufacturing Systems*, Vol 20, No 4, pp. 264-279.

Kutanoglu, E., and D., Wu, 2004, "Improving scheduling robustness via processing and dynamic adaptation", *IEE Transactions*, Vol 36, pp. 1107-1124.

Kutanoglu, E., and D., Wu, 2006, "Incentive compatible, collaborative production scheduling with simple communication among distributed agents", *International Journal of Production Research*, Vol 44, No 3, pp. 421-446.

Lawrence, S., and E., Sewell, 1997, "Heuristic, optimal, static and dynamic schedules when processing times are uncertain", *Journal of Operations Management*, Vol 15, pp. 71-82.

Liu, J., and K., Sycara, 1997, "Coordination of multiple agents for production management", *Annals of Operations Research*, Vol 75, pp. 235-289.

Panwalkar, S.S., and W., Iskander, 1977, "A Survey of Scheduling Rules", *Operations Research*, Vol 25, No 1, pp. 45-61.

Pendharkar, P.C., 1999, "A computational study on design and performance issues of multi-agent intelligent systems for dynamic scheduling environments", *Expert Systems with applications*, Vol 16, pp. 121-133.

Sabuncuoglu, I., and M., Bayız, 2000, "Analysis of reactive scheduling problems in a job shop environment", *European Journal of Operational Research*, Vol 126, pp 567-586.

Sabuncuoglu, I., and O., Kizilisik, 2003, "Reactive scheduling in a dynamic and stochastic FMS environment", *International Journal of Production Research*, Vol 41, No 17, pp. 4211-4231.

Shafaei, R., and P., Brunn, 1999, "Workshop scheduling using practical (inaccurate) data Part 1: The performance of heuristic scheduling rules in a dynamic job shop environment using a rolling time horizon approach", *International Journal of Production Research*, Vol 37, No 17, pp. 3913-3925.

Shaw, M.J., and A.B. Whinston, 1983, *Distributed planning in cellular flexible manufacturing systems, Technical Report, Management Information Research Center*, Purdue University.

Shaw, M.J., 1988, "Dynamic scheduling in cellular manufacturing systems: a framework for networked decision making", *JMS* Vol 7 No 2 , pp. 83–94.

Shen, W. and D. H., Norrie, 1999, "Agent based systems for intelligent manufacturing: a state of the art survey", *International Journal of Knowledge and Information Systems*, 1 (2), 129-156.

Shen, W., L., Wang, and Q., Hao, 2006a, "Agent-Based Distributed Manufacturing Process Planning and Scheduling: A State-of-the-Art Survey", *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, Vol 36, No 4, pp 563- 576.

Shen, W., Q., Hao, H.J., Yoon, and D.H., Norrie, 2006b, "Applications of agent-based systems in intelligent manufacturing: An updated review", *Advanced Engineering Informatics*, Vol 20, pp 415- 431.

Sozer, K., 2007, *State Based Modelling of Scheduling Agents in Intelligent Manufacturing*, M.S. Thesis, Bogazici University.

Tharumarajah, A., 2001, "Survey of resource allocation methods for distributed manufacturing systems", *Production Planning and Control*, Vol 12. No 1, pp. 58-68.

Tovar, E., and F., Vasques, 2001, "Distributed computing for the factory-floor: a real-time approach using WorldFIP networks", *Computers in Industry*, Vol 44. pp. 11-31.

Ouelhadj, D., 2003, *A Multi-Agent System for the Integrated Dynamic Scheduling of Steel Production*, PhD. Dissertation, The University of Nottingham.

Upton, D.M., 1998, *The operation of large computer controlled manufacturing systems*, Ph.D. Dissertation, Purdue University.

Vieira, G.E., J.W., Herrman, and E., Lin, 2003, "Rescheduling manufacturing systems: A framework of strategies, policies and methods", *Journal of Scheduling*, Vol 6, pp 39-62.

Wang, F., 2003, "Adaptive meeting scheduling for large-scale distributed groupware", *BT Technology Journal*, Vol 21, No 4, pp. 138-145.

Wang, Y.H., C.W., Yin, and Y., Zhang, 2003, "A multi-agent and distributed ruler based approach to production scheduling of agile manufacturing systems", *International Journal of Computer Integrated Manufacturing*, Vol 16, No 2, pp. 81-92.

Yen, B.P.C., 2002, "Communication infrastructure in distributed scheduling", *Computers & Industrial Engineering*, Vol 42, pp. 149-161.

Yu, S., 2006, "Priority-Based Event Message Scheduling in Distributed Virtual Environment", High Performance Computing and Communications, Lecture Notes in Computer Science, Vol 4208, pp 884- 893.

Zeng, Y., W., Cai, S. J., Turner, S., Zhou, and B., Lee, 2004, "Characterization and delivery of directly coupled causal messages in distributed systems", *Future Generation Computer Systems*, Vol 20. pp. 171-178.

REFERENCES NOT CITED

Al-Mouhamed, M., and H., Najjari, 2000, "Adaptive Scheduling of Computations and Communications on Distributed-Memory Systems", *Journal of Parallel and Distributed Computing*, Vol 60. pp. 716-740.

Bhat, P.B., V.K., Prasanna, and C.S., Raghavendra, 1999, "Adaptive Communication Algorithms for Distributed Heterogeneous Systems", *Journal of Parallel and Distribued Computing*, Vol 59. pp. 252-279.

Bernauer, M., and M., Schrefl, 2004, "Self-maintaining web pages: from theory to practice", *Data & Knowledge Engineering*, Vol 48. pp. 39-73.

Beynon, M.D., T., Kurc, U., Catalyurek, C., Chang, A., Sussman, and J., Saltz, 2001, "Distributed processing of very large datasets with DataCutter", *Parallel Computing*, Vol 27. pp. 1457-1478.

Bhat, P.B., C.S., Raghavendra, and V.K., Prasanna, 2003, "Efficient collective communication in distributed heterogeneous systems", *Journal of Parallel and Distributed Computing*, Vol 63. pp. 251-263.

Bilas, A., D., Jiang, and J.P., Singh, 2003, "Shared virtual memory clusters: bridging the cost-performance gap between SMPs and hardware DSM systems", *Journal of Parallel and Distributed Computing*, Vol 63. pp. 1257-1276.

Breitbart, Y., H.F., Korth, 1999, "Replication and Consistency in a Distributed Environment", *Journal of Computer and System Sciences*, Vol 59. pp. 29-69.

Cohen, E., H. Kaplan, and U., Zwick, 2003, "Connection caching: model and algorithms", *Journal of Computer and System Sciences*, Vol 67. pp. 92-126.

Cruz, J., and K., Park, 1999, "Toward Performance-Driven System Support for Distributed Clustered Environments", *Journal of Parallel and Distributed Computing*, Vol 59, pp. 132-154.

Dikaiakos, M.D., and D, Zeinalipour-Yazti, 2004, "A distributed middleware infrastructure for personalized services", *Computer Communications*, Vol 24. pp. 1464-1480.

Fei, Z., 2003, "A new consistency algorithm for dynamic documents in content distribution networks", *Journal of Parallel and Distributed Computing*, Vol. 63, pp. 916-926.

Floros, N., A.J.G., Hey, K.E., Meacham, J., Papay, and M,, Surridge, 1999, "Predictive resource management for meta-applications", *Future Generation Computer Systems*, Vol 15, pp. 723-734.

Friedman, R., R., Vitenberg, and G., Chockler, 2003, "On the composability of consistency conditions", *Information Processing Letters*, Vol 86. pp. 169-176.

Gou, L., P.B.,Luh, and Y., Kyoya, 1998, "Holonic manufacturing scheduling: achitecture, cooperation, mechanism, and implementation", *Computer in Industry*, Vol 37, pp. 213-231.

Helary, J.M., A., Mostefaoui, and M., Raynal, 2002, "Interval Consistency of Asynchronous Distributed Computations", *Journal of Computer and System Sciences*, Vol 64. pp. 329-349.

Hsieh, C., 2003, "Optimal task allocation and hardware redundancy policies in distributed computing systems", *European Journal of Operational Research*, Vol 147. pp. 430-447.

Hsieh, C., and Y., Hsieh, 2003, "Reliability and cost optimization in distribute computing systems", *Computers and Operations Research*, Vol 30. pp. 1103-1119.

Horowitz, K., and D., Malkhi, 2003, "Estimating network size from local information", *Information Processing Letters*, Vol 88. pp. 237-243.

Hwang, L., and C., Chang, 1999, "Analysis of a general limited scheduling mechanism for a distributed communication system", *Computer Networks*, Vol 31. pp. 1879-1889.

Joung, Y., 2001, "On Fairness Notions in Distributed Systems", *Information and Computation*, Vol 166, pp. 1-34.

Kaneda, K., K., Taura, and A., Yonezawa, 2003, "Virtual private grid: a command shell for utilizing hundreds of machines efficiently", *Future Generation Computer Systems*, Vol 19. pp. 563-573.

Lam, K., T., Kuo, W., Tsang, and G.C.K., Laq, 2000, "Concurrency Control in Mobile Distributed Real-Time Database Systems", *Information Systems*, Vol 25, No 4, pp. 261-286.

Maalouf, H.W., M.K., Gurcan, 2002, "Minimisation of the update response time in a distributed database system", *Performance Evaluation*, Vol 50. pp. 245-266.

Morin, C., P., Gallard, R., Lottiaux, and G., Vallee, 2004, "Towards and efficient single system image cluster operating system", Future Generation Computer Systems, Vol 20. pp. 505-521.

Nikolaidou, M., and D., Anagnostopoulos, 2003, "A distributed system simulation modelling approach", *Simulation Modelling Practice and Theory*, Vol 11. pp. 251-267.

Paul, S., Kaplan, and Z., Fei, 2001, "Distributed caching with centralized control", *Computer Communications*, Vol 24. pp. 256-268.

Park, G., 2004, "Performance evaluation of a list scheduling algorithm in distributed memory multiprocessor systems", *Future Generation Computer Systems*, Vol 20, pp. 249-256.

Sandholm, T.W., 2000, "Automated contracting in distributed manufacturing among independent companies", *Journal of Intelligent Manufacturing*, Vol 11, pp. 271-283.

Segre, A.M., S., Forman, G., Resta, and A., Wildenberg, 2002, "Nagging: A scalable fault-tolerant paradigm for distributed search", *Artificial Intelligence*, Vol 140. pp. 71-106.

Sohn, K., and S., Moon, 2000, "Achieving high degree of concurrency in multidatabase transaction scheduling: MTOS", *Journal of Systems Architecture*, Vol 46. pp. 687-698.

Soleimany, C., and S., Dandamundi, 2003, "Performance of a distributed architecture for query processing on workstation clusters", *Future Generation Computer Systems*, Vol 19. pp. 463-478.

Rezvan, M., K., Pawlikowski, and H., Sirisena, 2004, "A distributed cache architecture for QoS routing in large networks", *Computer Networks*, Vol 44. pp. 189-209.

Ulusoy, Ö., 1998, "Transaction processing in distributed active real-time database systems", *The Journal of Systems and Software*, Vol 42, pp. 247-262.

Veeravalli, B., and J., Yao, 2004, "Divisible load scheduling strategies on distributed multi-level treenetworks with communication delays and buffer constraints", *Computer Communications*, Vol 27, pp. 93-110.

Wang, C., and Y.M., Teo, 2001, "Supporting parallel computing on a distributed object architecture", *The Journal of Systems and Software*, Vol 56. pp. 261-278.

Weissman, J.B., L.R., Abburi, and D., England, 2003, "Integrated scheduling: the best of both worlds", Journal of Parallel and Distributed Computing, Vol 63. pp. 649-668.

Wu, K., and P.S., Yu, 2000, "Latency-sensitive hashing for collaborative Web caching", *Computer Networks*, Vol 33. pp. 633-644, 2000.

Yücesan, E., Y., Luo, C., Chen, I., Lee, 2001, "Distributed web-based simulation experiments for optimization", *Simulation Practice and Theory*, Vol 9. pp. 73-90.

Zhang, J., L., Gao, F.T.S., Chan, and P., Li, 2003, "A holonic architecture of the concurrent integrated process planning system", *Journal of Materials Processing Technology*, Vol 139. pp. 267-1119.