

Titre: Simulation and Reinforcement Learning-based Decision-making for Adaptive Production Management in Complex Manufacturing Systems
Title:

Auteur: Ahmed Sakr
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Sakr, A. (2023). Simulation and Reinforcement Learning-based Decision-making for Adaptive Production Management in Complex Manufacturing Systems [Thèse de doctorat, Polytechnique Montréal]. PolyPublie.
Citation: <https://publications.polymtl.ca/53417/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/53417/>
PolyPublie URL:

Directeurs de recherche: Soumaya Yacout, & Samuel Bassetto
Advisors:

Programme: Doctorat en génie industriel
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Simulation and Reinforcement Learning-based Decision-making for Adaptive
Production Management in Complex Manufacturing Systems**

AHMED SAKR

Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie industriel

Mai 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

**Simulation and Reinforcement Learning-based Decision-making for Adaptive
Production Management in Complex Manufacturing Systems**

présentée par **Ahmed SAKR**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Luc ADJENGUE, président

Soumaya YACOUT, membre et directrice de recherche

Samuel BASSETTO, membre et codirecteur de recherche

Amel JAOUA, membre

Amr ELTAWIL, membre externe

DEDICATION

Thank you, Almighty God, for your protection, guidance, and provision of strength, peace of mind, and health.

To my mother and my deceased father for their great role in my life and their numerous sacrifices for me.

To my beloved wife, *Eka*, who has been my source of moral and emotional support through my Ph.D. and M.Sc. as well. Thank you for your patience, your tolerance, and your appreciated comfort that you brought to my life.

To my beloved little angel, *Caira*, who appeared in my life and gave me a great push to finish my projects.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Soumaya Yacout. I have learnt a lot from her throughout my Ph.D. She engaged me in various activities that broadened my horizons and enriched my academic and research experiences. I believe that she gave me some valuable keywords that supported my research and opened many doors for creative ideas.

I would like to express my deep thanks to my co-supervisor, Samuel Bassetto, for his support, encouragement, and helpful suggestions. I appreciate his friendly and enjoyable environment at CIMAR-Lab and his creative way of doing research.

I am also thankful to Luc Adjengue, Amel Jaoua, Amr Eltawil, and Giovanni Beltrame for agreeing to attend my Ph.D. defence as jury members and representatives of the director of graduate studies.

I would like to express my deep appreciation to my colleagues for their help and support

RÉSUMÉ

Dans cette recherche, des agents d'apprentissage par renforcement (AR) sont formés pour faciliter la prise de décision dans des systèmes de production complexes. Les agents prennent en charge les tâches de répartition et de lancement des commandes, en tenant compte des indicateurs de performance clés du système. Prise de décision des agents les politiques sont formées dans un environnement simulé fourni par la simulation d'événements discrets. Ces agents sont appelés agents de gestion. Inspirés par le concept d'auto-jeu, des agents AR conflictuels et stimulants sont inclus. Les agents en conflit simulent véritables interruptions des ressources système. Ils interrompent les ressources qui devraient affecter de manière significative les performances du système. Les agents difficiles créent une variabilité de la demande. Ils sélectionnent des taux de demande qui défient le système et entraînent le non-respect des dates d'échéance. Le rôle des agents conflictuels et provocateurs est de défier activement les agents de gestion. Cela pousse les agents de gestion à explorer de nouvelles stratégies pour faire face à l'impact des interruptions et de la variabilité du taux de demande sur les performances du système. L'approche est appliquée à un cas d'utilisation pour un système de fabrication de semi-conducteurs réel.

La performance du système, employant des agents de gestion, est d'abord comparée à la répartition et à la libération des commandes basées sur l'heuristique traditionnellement appliquées. La performance est évaluée à l'aide de paramètres comprenant, mais sans s'y limiter, le débit et le pourcentage de déchets. Ensuite, leurs performances sont testées dans deux cas : avec et sans la présence des agents conflictuels. Enfin, les agents de gestion sont testés en présence d'agents à la fois conflictuels et provocateurs. Dans ce cas, le taux de demande varie pour le même travail en cours, créant une variabilité au sein du système. L'application de AR pour la gestion des agents montre des résultats prometteurs par rapport aux heuristiques traditionnellement appliquées. La première comparaison montre une amélioration du débit de production, une réduction du temps total sans valeur ajoutée et une réduction des rebuts. La deuxième comparaison montre une amélioration des politiques apprises des agents de gestion. L'amélioration a été obtenue dans le même temps de formation pour les agents de gestion sans interruption, grâce à la présence d'agents conflictuels. Le dernier test pour les agents, dans lequel des agents conflictuels et provocateurs étaient présents, montre que les agents de gestion gèrent le taux de demande variable sans entraîner de retards excessifs dans l'exécution des ordres de travail. Les agents conflictuels et stimulants ont amélioré les politiques apprises des agents, malgré la variabilité causée par les opérateurs et les demandes variables des clients.

ABSTRACT

In this research, Reinforcement Learning (RL) agents are trained to facilitate the decision-making in complex production systems. The agents support dispatching and order release tasks, considering the system's Key Performance Indicators (KPIs). Agents' decision-making policies are trained within a simulated environment provided through Discrete-Event Simulation (DES). These agents are referred to as the managing agents. Inspired by the self-play concept, conflicting and challenging RL agents are included. The conflicting agents simulate real system resources interruptions. They interrupt the resources that are expected to significantly affect the system's performance. The challenging agents create demand variability. They select demand rates that challenge the system and cause due dates to be missed. The role of both conflicting and challenging agents is to actively challenge the managing agents. This drives the managing agents to explore new strategies for tackling the impact of interruptions and demand rate variability on the system's performance. The approach is applied to a use case for a real semiconductor manufacturing system.

The performance of the system that employs managing agents is first compared to traditionally applied heuristic-based dispatching and order release system. The performance is assessed through metrics including, but not limited to, throughput and waste percentage. Then, their performance is tested in two cases: with and without the conflicting agents' presence. Finally, the managing agents are tested in the presence of both conflicting and challenging agents. In that case, the demand rate is varied for the same Work-In-Process (WIP) creating variability within the system. The application of RL for managing agents shows promising results over traditionally applied heuristics. The first comparison shows an improvement in the production throughput, a reduction in the total non value-added time, and a reduction in scrap. The second comparison shows an improvement in the learnt policies of the managing agents. This improvement was achieved through the conflicting agents' presence. This was achieved within the same training time for managing agents with no interruptions. The last test for the agents, in which both conflicting and challenging agents were present, shows that the managing agents handle the varying demand rate without resulting in excessive delays in fulfilling job orders. The introduction of conflicting and challenging agents improved the management agents' learnt policies. This promotes the applicability of such RL-based decision-making in real systems considering the variability caused by human operators and variable customer demands.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF TABLES	xii
LIST OF FIGURES	xiii
LIST OF SYMBOLS AND ACRONYMS	xvii
LIST OF APPENDICES	xviii
CHAPTER 1 INTRODUCTION	1
1.1 Basic concepts	1
1.1.1 The Evolution of Production Management Approaches	1
1.1.2 Complex Production Systems	2
1.1.3 The Industry 4.0 Paradigm	2
1.1.4 Simulation and reinforcement learning	3
1.2 Elements of the problem	4
1.3 Research objectives	5
1.4 Research questions	6
1.5 Research hypotheses	6
1.6 Points of originality	7
CHAPTER 2 THESIS ORGANIZATION	8
CHAPTER 3 LITERATURE REVIEW	12
3.1 Discrete-Event Simulation in production management	12
3.2 Digital Twin-based applications in industry	14
3.3 Decision-making decentralization and holonic manufacturing	16

3.4	Reinforcement Learning applications in industry	17
CHAPTER 4 ARTICLE 1: A DISCRETE EVENT SIMULATION LOGIC FOR SEMI- CONDUCTOR PRODUCTION PLANNING AND CONTROL WITHIN INDUS- TRY 4.0 PARADIGM		
4.1	Abstract	22
4.2	Introduction	23
4.3	Problem statement	25
4.3.1	Semiconductor industry background	25
4.3.2	Process description and sources of complexity	25
4.4	Methodology	26
4.4.1	Selecting the appropriate simulation modeling type	26
4.4.2	Data-driven agent-based modeling	27
4.4.3	Input data structuring	28
4.4.4	Simulation logic	30
4.5	Goals	34
4.6	Verification and validation	35
4.7	Conclusion and future works	36
CHAPTER 5 ARTICLE 2: BUILDING DISCRETE-EVENT SIMULATION FOR DIG- ITAL TWIN APPLICATIONS IN PRODUCTION SYSTEMS		
5.1	Abstract	37
5.2	Introduction	38
5.3	Background	40
5.3.1	DES of Production Systems	40
5.3.2	The Digital Twin (DT) Technology in Industry 4.0	41
5.3.3	Role of DESs in DTs	41
5.4	DT Requirements	42
5.5	Limitations of Traditional DES Integration	43
5.6	Proposed Approach	44
5.6.1	Semiconductor Manufacturing Systems	44
5.6.2	DES Modeling Scheme	45
5.6.3	Data-Driven Modeling Implementation	45
5.6.4	Agent-Based Modeling Implementation	46
5.6.5	Integration Scripts	48
5.7	Use Case Experiment	54
5.8	Conclusion	55

CHAPTER 6	ARTICLE 3: SIMULATION AND DEEP REINFORCEMENT LEARNING FOR ADAPTIVE DISPATCHING IN SEMICONDUCTOR MANUFACTURING SYSTEMS	57
6.1	Abstract	58
6.2	Introduction	58
6.3	Literature review	60
6.3.1	Optimization with simulation models	60
6.3.2	Reinforcement learning with simulation models	61
6.4	Problem statement	62
6.4.1	Semiconductor industry overview	62
6.4.2	Semiconductor processing aspects	63
6.4.3	Dispatching tasks in semiconductor industry	64
6.5	Methodology	64
6.5.1	System modeling overview	64
6.5.2	Simulation model in depth	65
6.5.3	Deep-Q-Network	69
6.5.4	Deep-Q-Network (DQN) for dispatching and resource allocation . . .	70
6.5.5	DQN implementation	76
6.6	Results and discussion	76
6.6.1	Case study description	76
6.6.2	DQN configuration	78
6.6.3	Results and comparison	78
6.6.4	Discussion for DQN agents' performance	84
6.7	Conclusion	85
CHAPTER 7	ARTICLE 4: DEEP REINFORCEMENT LEARNING FOR ADAPTIVE DISPATCHING AND ORDER RELEASE CONSIDERING CONFLICTING HUMAN AND EQUIPMENT INTERRUPTIONS	87
7.1	Abstract	88
7.2	Introduction	88
7.3	Literature review	91
7.3.1	Reinforcement learning for decision making	91
7.3.2	Human operators in Industry 4.0	92
7.3.3	Conflicting RL for improving learnt policies	93
7.4	Problem statement	93
7.4.1	Reinforcement learning-based decision making	93

7.4.2	Outlines of semiconductor industry	94
7.4.3	Within products' process flow	95
7.4.4	Management challenges	95
7.5	Methodology	96
7.5.1	Model building through DES	97
7.5.2	Data-driven modeling	97
7.5.3	Agent-based modeling	99
7.5.4	DQN agents	101
7.5.5	Order release agents	102
7.5.6	Dispatching agents	106
7.5.7	Interruption agents	109
7.6	Results and discussion	113
7.6.1	Use case overview	113
7.6.2	Agents DQN configuration	115
7.6.3	Results	115
7.7	Conclusion	123
CHAPTER 8 ARTICLE 5: ADAPTIVE DISPATCHING AND ORDER RELEASE THROUGH SELF-PLAY DEEP REINFORCEMENT LEARNING CONSIDERING DYNAMIC DEMAND AND PRODUCTION CONDITIONS		
8.1	Abstract	125
8.2	Introduction	125
8.3	Background	127
8.4	Problem description	129
8.4.1	Problems of randomly assigned demand	129
8.4.2	Self-play implementation challenges	130
8.5	Methodology	130
8.5.1	DQN agents within DES	130
8.5.2	Overview for agents within the DES model	131
8.5.3	Managing agents: dispatching and order release agents	132
8.5.4	Challenging agents: customers	136
8.5.5	Conflicting agents: operators and equipment interrupting agents	138
8.6	Results and discussion	139
8.6.1	DQN configuration for agents	140
8.6.2	Training parameters	140
8.6.3	Actions	140

8.6.4 Results	141
8.7 Conclusion	149
CHAPTER 9 GENERAL DISCUSSION	151
9.1 Summary	151
9.2 Future Research	154
CHAPTER 10 CONCLUSION AND RECOMMENDATIONS	155
REFERENCES	158
APPENDICES	170

LIST OF TABLES

Table 6.1	DQN's parameters for dispatching agents	79
Table 6.2	Learning parameters for dispatching agents	79
Table 7.1	Operator interruption events	110
Table 7.2	DQN's parameters for managing and conflicting agents	116
Table 7.3	Learning parameters for managing and conflicting agents	116
Table 8.1	Agents' learning parameters	141
Table 8.2	DQNs' parameters for all agents	142
Table 8.3	Customer agents actions	142

LIST OF FIGURES

Figure 1.1	Different types of system modelling	3
Figure 1.2	The agent-environment interaction in RL	4
Figure 2.1	Thesis Organization	9
Figure 4.1	Different types of system modelling	27
Figure 4.2	Working logic of equipment stations' agents	30
Figure 4.3	Working logic of equipment uptime/downtime agents	31
Figure 4.4	Working logic of operators' activity/interruption agents	32
Figure 4.5	Lot creation and assignment of important information logic	32
Figure 4.6	Lot processing logic	32
Figure 4.7	The after-processing logic	34
Figure 5.1	Digital Twin relationship with its corresponding real system	43
Figure 5.2	Data structure of a data-driven agent-based DES model for a semiconductor manufacturing system	47
Figure 5.3	Agents operating logic through the DES model	49
Figure 5.4	Introduction of the integration library within a Digital Twin	50
Figure 5.5	DES model reconfiguration steps	50
Figure 5.6	DES model's state initialization steps	53
Figure 5.7	Average queue waiting time at each equipment with and without state initialization	55
Figure 5.8	Average total time in system and average WIP for products with and without state initialization	55
Figure 6.1	Data flow in data-driven DES	67
Figure 6.2	Products-related sub-models overview	68
Figure 6.3	Dispatching agents learning process	69
Figure 6.4	Agents learning process through DQN algorithm and its virtual environment	71
Figure 6.5	Dispatching agents through the simulation model	72
Figure 6.6	CTX in system using heuristics	80
Figure 6.7	CTX in system using DQN	80
Figure 6.8	Timers rework% and junk% in system using heuristics	81
Figure 6.9	Timers rework% and junk% in system using DQN	81
Figure 6.10	Timers rework% and junk% in system without hold-in-advance	82
Figure 6.11	Timers rework% and junk% in system using FIFO only	82

Figure 6.12	Throughput of system using heuristics	83
Figure 6.13	Throughput of system using DQN	83
Figure 6.14	Throughput of system using heuristics without hold-in-advance	83
Figure 6.15	Throughput of system using FIFO only	84
Figure 7.1	Data categories and relations of the proposed data-driven modeling for a semiconductor manufacturing system	99
Figure 7.2	Agents through the proposed agent-based modeling for a semiconductor manufacturing system	100
Figure 7.3	Agents Learning process through RL	101
Figure 7.4	Agents learning process applying DQN for RL	102
Figure 7.5	Working logic of order release agent for product type 1	103
Figure 7.6	Working logic of dispatching agent for equipment type 3 station	106
Figure 7.7	Operator interruption agent working logic	110
Figure 7.8	Equipment interruption agent working logic	111
Figure 7.9	Order release times in normal system applying interarrival time distributions	117
Figure 7.10	Order release times in system with DQN managing and conflicting agents	117
Figure 7.11	CTX in normal system applying heuristics and random operator/equipment interruptions	118
Figure 7.12	CTX in system with DQN managing and conflicting agents	118
Figure 7.13	Timers rework% and junk% in system applying heuristics and random operator/equipment interruptions	120
Figure 7.14	Timers rework% and junk% in system with DQN dispatching agents only and no interruptions	120
Figure 7.15	Timers rework% and junk% in system with DQN managing and conflicting agents	121
Figure 7.16	Throughput of system applying heuristics and random operator/equipment interruptions	121
Figure 7.17	Throughput of system with DQN dispatching agents only and no interruptions	121
Figure 7.18	Throughput of system with DQN managing and conflicting agents ..	122
Figure 7.19	Job orders' relative earliness/tardiness in system applying heuristics and random operator/equipment interruptions	122
Figure 7.20	Job orders' relative earliness/tardiness in system with DQN managing and conflicting agents	122

Figure 8.1	RL agents learning process within the DES model [1]	131
Figure 8.2	Functionalities of the agents within the DES model	133
Figure 8.3	Customer agents reward value with respect to job completion time ...	138
Figure 8.4	Release times for lots in system applying RL for dispatching agents only	142
Figure 8.5	Release times for lots in system applying RL for managing, conflicting, and challenging agents	143
Figure 8.6	Total released lots with only RL dispatching agents and with RL for all agents	143
Figure 8.7	Box-plot for the available times for job orders set by customer agents	144
Figure 8.8	CTX in system applying RL for dispatching agents only	144
Figure 8.9	CTX in system applying RL for managing, conflicting, and challenging agents	145
Figure 8.10	Percentages of rework and junk of timers in system applying RL for dis- patching agents only	145
Figure 8.11	Percentages of rework and junk of timers in system applying RL for man- aging, conflicting, and challenging agents	146
Figure 8.12	Absolute slack time for jobs upon completion	146
Figure 8.13	Box-plot for absolute slack time for jobs upon completion	147
Figure 8.14	Slack to available time ratio for jobs upon completion	147
Figure 8.15	Box-plot for slack to available time ratio for jobs upon completion ..	148
Figure A.1	The data structure of the simulation model	170
Figure A.2	The agents working logic within the simulation model	171
Figure A.3	The process flow of one of the use case products	173
Figure A.4	Sample of the timers information	174
Figure A.5	Sample of the rework and IF conditions information	174
Figure A.6	Sample of the recipes that are considered for products	176
Figure A.7	Sample of the dedication of certain equipment for recipes	177
Figure A.8	Sample of the equipment information	178
Figure A.9	Sample of the equipment included within each equipment type	179
Figure A.10	Sample of the Operators information	180
Figure A.11	Sample of the Operators information for shifts	181
Figure A.12	Sample of the Operators information for qualifications	182
Figure A.13	Customer agents working logic	183
Figure A.14	Dispatching agents creation	184
Figure A.15	Interruption agents creation and working logic	185

Figure A.16	Equipment uptime/downtime agents creation and working logic	186
Figure A.17	Moving averages calculation of for agents' reward	187
Figure A.18	SR calculation for each product type	188
Figure A.19	Order release of products	189
Figure A.20	Preparing lots to enter equipment stations	190
Figure A.21	Sending lots to enter equipment stations	191
Figure A.22	Dispatching for equipment: Dispatching agents logic	192
Figure A.23	Dispatching for equipment: Dispatched lots	193
Figure A.24	Dispatching for equipment within clusters: included equipment types	194
Figure A.25	Dispatching for equipment within clusters: main dispatching	195
Figure A.26	Dispatching for equipment within clusters: routing dispatched lots .	196
Figure A.27	Seizing resources	198
Figure A.28	Delaying for processing	199
Figure A.29	Unloading equipment	200
Figure A.30	Recipe yield assignment	201
Figure A.31	Rework/Rejoin	202
Figure A.32	IF Conditions and assignment of the next step	203
Figure A.33	Transferring lots to their next step	204
Figure A.34	Transferring lots to their next step	205
Figure A.35	Check for timers steps	206
Figure A.36	Define equipment for timers steps	207
Figure A.37	Check for timers' criticality	208
Figure A.38	High timer's criticality: seize in advance	210
Figure A.39	Low timer's criticality: check availabiltiy in advance	211
Figure A.40	Seizing equipment within timers	212
Figure A.41	Exceeding timers' limit	213
Figure A.42	Exceeding timers' limit	214

LIST OF SYMBOLS AND ACRONYMS

AI	Artificial Intelligence
CLRP	Closed-loop Release Policies
CONWIP	CONstant Work in Process
CPS	Cyber-Physical System
CPS	Cyber-Physical System
CTX	Cycle Time X factor
DES	Discrete-Event Simulation
DQN	Deep-Q-Network
DS	Digital Shadow
DT	Digital Twin
EDD	Earliest Due-date
FIFO	First-In-First-Out
I4.0	Industry 4.0
IoT	Internet of Things
KPI	Key Performance Indicator
ML	Machine Learning
OLRP	Open-loop Release Policies
RL	Reinforcement Learning
SPT	Shortest Processing Time
SR	Slack Ratio
VBA	Visual Basic for Applications
WIP	Work-In-Process
WR	Workload Regulation

LIST OF APPENDICES

Appendix A	The Simulation model	170
------------	--------------------------------	-----

CHAPTER 1 INTRODUCTION

Advancements in digital technologies, which are adopted by the Industry 4.0 (I4.0), have created opportunities and challenges for production management systems. Communication and information technologies, based on Internet of Things (IoT) concepts, help interconnecting shop floor components for better management and control. This allows communicating integrated systems to achieve certain tasks or to improve their performance. Similarly, the same concept is industrially applied, which gives an opportunity to remotely monitor system components and devices/equipment. This allows for physical-virtual integrity reaching reliable decision-making support tools. Hence, achieving efficient and profitable production strategies [2, 3].

One of the reasons for the current need of flexible manufacturing systems is the fast-growing competition. Such competition requires adaptability to changes by manufacturers. Flexible manufacturing allows a level of adaptability, providing capabilities to react to changes, whether predictable or unpredictable [4]. Through I4.0, embedded Artificial Intelligence (AI) provides reliable and rapid decision-making and improve real systems' performance. In this context, researchers addressed the application of these tools in manufacturing systems. The general objective is to overcome challenges that are related to production planning, management, and control, while using the connectivity that is provided through IoT. The application of Machine Learning (ML)-based decision-making has been recognized by many researchers as a promising approach [5]. It provides the required intelligence for decision-making considering the system's state.

1.1 Basic concepts

1.1.1 The Evolution of Production Management Approaches

Manufacturers seek for improving the production efficiency [6]. To achieve these goals, they invest in developing their management and control strategies. This is challenged by production costs that normally increase if a management strategy is not properly selected or applied.

The I4.0 was sought to implement decision-making intelligence for the industry in 2013 [7]. This intelligence is provided through the internet connection, AI, and ML applications. There is a continuous evolution of production management and control methods. This evolution

extends from the automation era to the networking one. The management intelligence became a crucial requirement for manufacturers to maintain their competence [6]. Accordingly, management and control strategies shifted from being reactive to predictive and proactive.

1.1.2 Complex Production Systems

A product or production is considered to be complex if it is subject to variable customers' demands, and/or if its production flow is not fixed. The enterprises have been exposed to this type of production [8]. This is caused by the increasing global competition and the shift towards customer-centric markets. To increase customer satisfaction, internal logistic flexibility is required. Internal logistics' quality can be expressed by certain performance indicators such as the average throughput or order's tardiness. For leveraging the competitiveness, low logistic costs must be considered along with improving its quality [9].

1.1.3 The Industry 4.0 Paradigm

Industry 4.0 is a digital revolution that is a byproduct of the advancement in information technology, the connectivity to the internet, and the sensor technology [10]. This indicates future transformation of current production systems including their management strategies [11].

The IoT is the technological basis of the I4.0 [12]. It allows connection, exchange, and collection of data between various devices, or things, through the internet. This physical-virtual integration facilitates the implementation of quick and reliable decision-making tools. Thus, providing efficient and profitable production [2].

Cyber-Physical System (CPS) is one of the key technologies in I4.0. In a CPS, the physical world, such as a production system, and the digitized cyber-world are combined through the communication technologies. This is the cornerstone of I4.0. It provides a powerful and rapid decision-making for productivity improvement. CPSs have embedded intelligence that is provided through data models [3,13]. The intelligence is important for data-to-information conversion. It includes several methodologies such as diagnosis and prognosis algorithms and health management applications. ML is one of the methods used for providing the required intelligence for decision-making. It is used for providing adaptive decisions within manufacturing systems [14].

Within a CPS, Digital Twins (DTs) are virtual representation or simulation of physical systems. They are used to simulate reality for different purposes. With the aid of real-time synchronization of data, such simulations are able to represent current state of its real system [2,15]. DTs enable simulation to optimize and control production lines continuously,

based on real context instead of estimated or historical data as input [16].

1.1.4 Simulation and reinforcement learning

Simulation is a methodology that mimics the behavior of a real system. This includes natural phenomena, machine processes, or a manufacturing/service system. In manufacturing, simulation models are employed for planning through system performance prediction. Thus, different production scenario are assessed through experiments using these models [17]. There are various types of system modelling tools, including Discrete-Event Simulation (DES), according to the system behavior whether it is static/dynamic and deterministic/stochastic as shown in Fig. 1.1.

DES models evolve in a time-dependent manner. They advance in discrete unequal steps of time. After finishing all events of the current step, it advances to the next earliest event starting time [18]. In this type of simulation, the real system is represented by a set of entities and resources. Entities move through the system model, such as products. Resources, having certain capacity, provide services to the entities. The system state information is derived from attributes of each entity, and from global variables. It is updated upon advancing in time [18, 19].

Within DES, agent-based modeling has the advantage of employing agents that check various aspects and make decisions accordingly. The term "agent" describes a software component that perceives, acts, and communicates to complete certain tasks. They are desired for addressing different properties such as autonomy and responsiveness. Traditionally, they work according to predefined logic. However, they provide autonomy upon employing algorithms

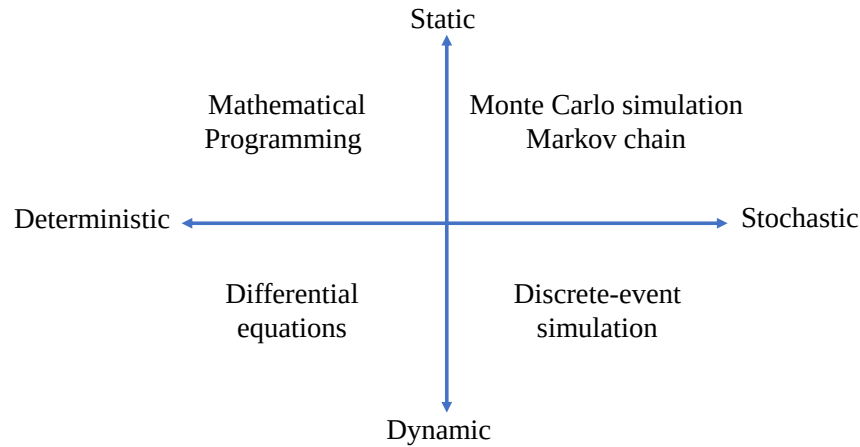


Figure 1.1 Different types of system modelling

such as ML [20].

Reinforcement Learning (RL) is one of the branches of ML. In RL, agents are trained within an environment such as a DES model [21, 22]. It has to take actions that are expected to yield the best results. It emulates the process of how humans learn [23]. RL is most suitable when there is no knowledgeable supervisor. Thus, the agent needs to learn from interaction with its environment as shown in Fig. 1.2. This is achieved through exploiting its experience and exploring new strategies [21].

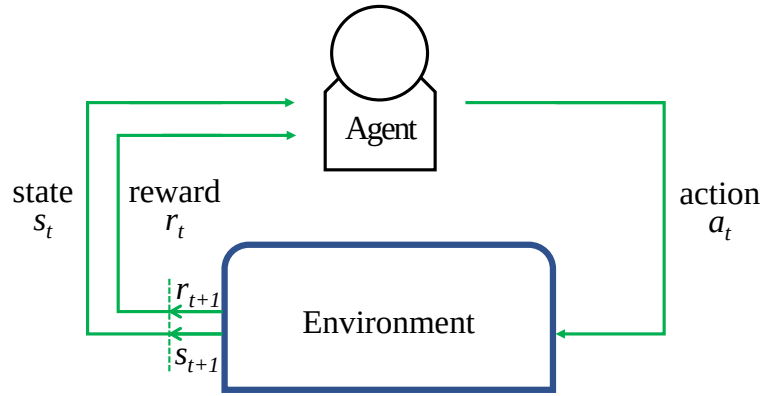


Figure 1.2 The agent-environment interaction in RL

1.2 Elements of the problem

Recent growth of digital technologies, including communication and information technologies, are capable of interconnecting shop floor resources. This new era adopts new concepts, such as decision-making decentralization, that have become a strategic response by competitive manufacturers [2, 3]. The connected machines perform continuous data collection regarding performance, maintenance, and any other aspects. This provides opportunities for identifying patterns and insight through the analysis of these data. Normally, such tasks are hard for a human to do in a reasonable duration [24].

ML techniques are used to extract information from data and support decision-making. RL is one of these techniques that has been utilized for such tasks. It involves training artificial agents to take action to increase their rewards within their environment. For production systems, the environment for these agents is provided through DES. The simulation generates states, apply the agents' actions, and accordingly assigns reward to them corresponding to actions' adequacy.

Employing artificial agents for decision-making is challenged by the level of intelligence they can reach. This level is limited by system's states that they experience through training. DES may randomly simulate system disturbances such as operators' interruptions and equipment loss of efficiency. This does not necessarily pose challenges as the selected operator or equipment may not cause significant disturbances as other ones. The concept of on-line learning has been suggested in some researches as a continuation of off-line learning. It could enhance the readiness of the agents to offer solutions under unusual disturbances. However, relying on this strategy makes agents vulnerable to such events. This is due to the fact that the learning process for these agents takes hundreds of thousands to reach adequate performance. Thus, upon the occurrence of such unusual events, they may perform poorly.

1.3 Research objectives

The objective of this research is to develop a decision-making support tool for complex production systems. The research addresses simulating the real system through DES considering the DT concepts within the I4.0 paradigm. This achieves model flexibility to adopt complex systems behaviors. Further, the research establishes RL-based agents to achieve an embedded intelligence through decision-making agents. Managing agents are trained to make appropriate decisions by interacting with the DES which provides a training environment. They make decisions for dispatching and order release tasks. The state information and reward functions are formulated such that cooperation is achieved among agents. Inspired by the self-play concept, the managing agents are challenged by other conflicting and challenging agents, which seek conflicting objectives. This is proposed to improve the agents' learnt policies considering human presence, normal resources interruptions, and demand variability in real systems. This training strategy is proposed to avoid prolonging training periods and their associated problems in RL. The objectives of this research are categorized as follows:

Objective A: Develop a DES model considering DT concepts.

Develop a DES for our use case manufacturing system by applying the Data-driven and Agent-based concepts for model building. This includes structuring the input data that the model will acquire throughout its execution. The Data-driven model building ensures flexibility to adopt complex systems behavior. The Agent-based modeling provides the required functionalities for agents. It allows the model to acquire decisions made by RL agents, and to provide the training environment for them.

Objective B: Apply RL for decision-making and achieve cooperation among the managing agents.

Create managing agents for dispatching and order release decisions. This adopts the concept of decision-making decentralization. The training is achieved through DES and the reward is calculated according to the general production performance. The performance is measured by different Key Performance Indicators (KPIs) including cycle time, resource utilization, and junk or waste reduction. State and reward are formulated to promote cooperation and avoid focus on localized objectives.

Objective C: Develop a self-play approach to employ real systems variabilities to improve the learnt policies of the managing agents.

The research considers real systems variabilities such as human operator existence and resources interruption. It also addresses demand variation. Conflicting and challenging agents, applying RL, are introduced to create these variabilities within the simulation. This is inspired by the self-play concept that is applied in computer games. Thus, managing agents, as well as the other agents, are encouraged to explore new strategies and overcome the effects of one another. Thus, agents learn how to treat different unusual situations in a real system. It improves the learnt policies without prolonging training time.

1.4 Research questions

This research is centered on the following questions:

- The agents apply Deep-Q-Network (DQN) for RL so that they replace the traditional heuristics. How can agents be trained to improve the general performance given the system complexity?
- Given decision-making decentralization, do agents learn to cooperate with each other and avoid focusing on local objectives?
- How to apply the self-play concept to a production system so that it represents variabilities that exist in real systems?
- Do challenging agents encourage managing agents to improve their learnt policies?
- What are the limitations for application of this approach to real systems?

1.5 Research hypotheses

The approach in this research is based on the following hypotheses:

- Recent researches show promising results for the application of multiple RL agents. This demonstrates success in replacing the traditional heuristics. The DQN showed a good training strategy for the agents. This resulted in improved performance for manufacturing systems.

We assume that applying this strategy leads to appropriate learning for the agents.

- The concept of introducing conflicting and challenging agents is inspired by the self-play that is applied in computer games. An example for this concept is given by [25, 26]. Through this approach, the agents' performance reaches superhuman levels with no need for human knowledge in training.

Based on this success in computer games, the proposed approach of the conflicting and challenging agents improves the managing agents' learnt policies. This avoids the need for prolonging training periods which may not lead to performance improvement.

1.6 Points of originality

- Proposing a comprehensive approach to apply RL-based decision-making in semiconductor manufacturing systems through DES and DQN. This is achieved through multiple agents that learn independently adopting decision-making decentralization.
- Formulating state information and reward function that support cooperative behavior among agents. This is done considering specific real production challenges that require direct cooperation among agents to tackle.
- Considering real systems variabilities caused by resources interruptions during training. We increase their significance, within the limited training time, through the proposed self-play scheme for conflicting agents.
- Training managing agents for handling less frequent events, mainly system resources interruptions, and employing them to improve the learnt policies without prolonging training period. The proposed approach involves conflicting agents that apply the concept of self-play in a production system context.
- Introducing a self-play scheme between the system and customers that vary their demand rate. This trains the managing agents for handling variable customer demands in an adaptive way according to their learnt policies.

CHAPTER 2 THESIS ORGANIZATION

This research is divided into 4 general elements, as shown in Fig. 2.1. These elements cover building the DES model, implementing the RL, considering a self-play scheme for system interruptions, and introducing a game-like self-play scheme between the system and the customers that control the demand rates.

In the first part, the DES model is established for the semiconductor use case. In Chapter 4, model logic is introduced to achieve DES for the use case. The logic includes the products' variety, and the specific process flow for each product type. This includes preprocessing and postprocessing steps that take place before and after each processing step in the real system. The logic is based on data-driven and agent-based modeling to provide the flexibility required for modeling such complex systems. In addition, the agent-based modeling support applying any logic within the agents that are responsible for decisions. In Chapter 5, the DES model is developed considering DT concepts. This was done as such complex systems are subject to changes that need to be reflected in the model. In addition, as a decision-making support tool, some decisions are required in real-time. Hence, such DES will need to reside within a DT sandbox. For adopting decision-making agents within the model, the agents' submodels were developed such that they could perform their functionalities through preprogrammed logic, applying heuristics, for instance, or through RL algorithms. The DES model was developed, including two extra features through an integration layer. These features are model reconfiguration and state initiation.

In the second part, Chapter 6, RL was utilized to replace heuristics for dispatching agents. This was done through an individual DQN for each dispatching agent to achieve a decentralized scheme for decision-making. A dispatching agent was assigned for each equipment station within the model. The link between the DES model, through Arena, and the RL algorithm, through Python, was developed with the aid of Visual Basic for Applications (VBA). The state information was formed to achieve a comprehensive representation to support agent's training. It included local information for the agent's station and global information for the rest of the system to support cooperation among agents. Likewise, the reward function was formulated to reach adequate weight for local and global objectives. The criticality of some reward terms was shown in specific situations that involve decisions from multiple agents, up to three agents in our specific case. A special formulation was developed for these terms that assigns adequate reward for each agent according to their relative responsibility within such situations. This avoided having trivial actions when a fixed reward value was assigned,

as traditionally done, for all associated agents. A comparison was performed to test the proposed approach and compare it to the applied heuristic in the use case. It was shown the improved performance of the system when RL is applied for dispatching agents.

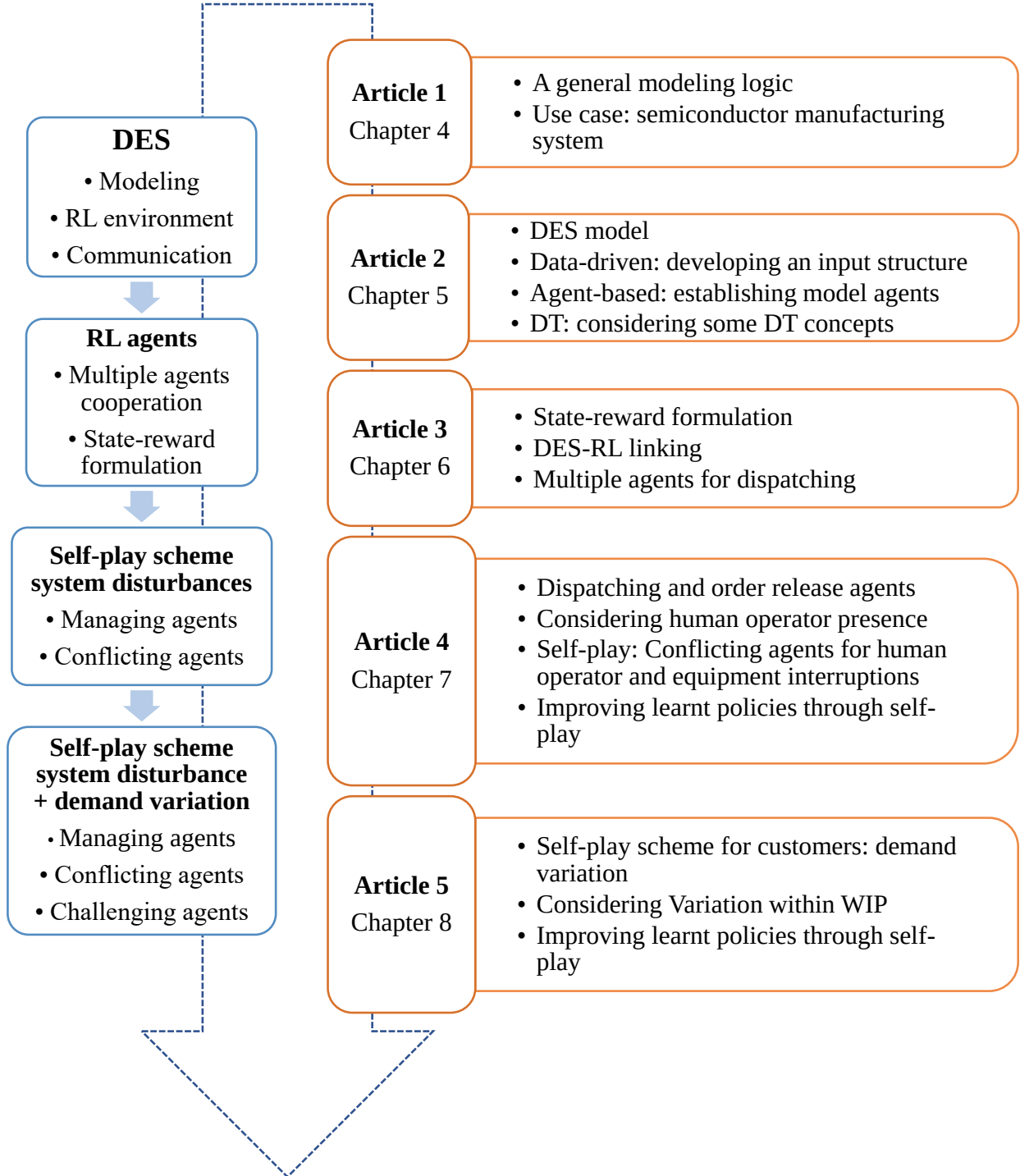


Figure 2.1 Thesis Organization

As the research progressed, the need to include some aspects of the real system was considered. These aspects included the presence of human operators and equipment interruptions that take place during processing the products. In Chapter 7, the objective was to train and test the agents' performance when these types of interruptions are present. The basic method was to add such interruptions randomly within the simulation. However, such randomly assigned interruptions do not consistently impact the system's performance significantly. In RL context, such less-frequent events slow down the learning convergence and add high variance. Hence, even prolonging training times does not solve this problem, in addition to being impractical. Inspired by the self-play concept used in computer games for agents, the same concept was used to create two extra agents to select and interrupt the operator or equipment that is expected to significantly impact system performance. We referred to them as the conflicting agents. An extra agent type was introduced for order release. An order release agent was assigned for each product type to decide when to start the next lot of wafers. We referred to the dispatching and order release agents as the managing agents. The results showed an improvement in the general performance of the dispatching agents as compared to their previous performance when there was no interruption. This improvement was shown specifically in the reduced percentages of rework and junk. This was achieved even though the order release pushed the system with more lots than it had with no interruptions. This showed the cooperation between the managing agents, and their readiness to tackle the conflicting agent impact simulating system interruptions. This improvement was achieved without prolonging the training time.

Due to the success of the self-play scheme implementation for system interruptions, we extended its application to include customers as well. In such a manufacturing system, and due to the move towards customer-centric production, the demand rates vary. This exposes the system to disturbances within the Work-In-Process (WIP) leading to congestion and tardy deliveries. The objective in Chapter 8 was to establish customer agents that challenge the system with their demand rates. This was done in a game-like scheme in which customers played the role of opponents which challenged the system. For the managing agents, this trains them for varying demand rates that target their learnt policies adaptively. The reward for customer agent was formulated so that it targeted achieving a certain level of job orders tardiness. Their reward was bounded so that they do not select actions that create inevitable situations for the system. The results showed that the system managed to mostly fulfil job orders with no tardiness. As compared to the previous system with only dispatching agents and no interruptions or variable demand, the dispatching agents, having the same DQN configurations, learnt better policies when conflicting and challenging agents were present. This result was achieved even though the WIP state varied according to the demand rate. One

of the sources of WIP variation was the changing Slack Ratio (SR) for each product types, which relates the remaining time till the due date to the remaining quantity to be produced for each job order.

CHAPTER 3 LITERATURE REVIEW

3.1 Discrete-Event Simulation in production management

DES has been identified as a method for analysis and improvement of manufacturing systems performance. It is used to improve the performance of complex manufacturing systems [27]. DES answers key operational questions for improving system's KPIs such as throughput, resource allocation, and resource utilization. This enables decision-makers to apply beneficial operational modifications. Hence, it enhances their production performance and helps reaching their targets [28].

In this context, Reményi and Staudacher [29] presented a systematic simulation-based approach for the identification of a suitable scheduling rules application. They addressed the problems of production challenges of the Maintenance, Repair, and Overhaul sector of the aircraft industry. Specifically, cost prediction and turnaround time. Ghani et al. [30] used DES for modeling a production assembly line in the automotive industry. They developed an algorithm for energy usage optimization based on dynamic property adjustments such as accelerations and motion types. In their research, the DES model was used to select different variables that are related to the production performance. Production rate, cycle time, and shift patterns were optimized along with energy consumption minimization. McDonald et al. [31] addressed the effect of varying demand levels on a motor manufacturing cell. They developed a DES using Arena simulation to determine production capabilities and potential problems for various demand levels. Their results indicated the potential bottlenecks for each level of demand. Korytkowski et al. [32] optimized the dispatching rules for a commercial printing production system. They applied simulation-based heuristics to optimize dispatching rule allocation. Their DES model was optimized through a genetic algorithm. They improved the cycle time,] and reduced orders tardiness and queues size.

Cerna et al. [33] addressed the problem of project delays in underground mining operations that are related to muck removal planning. They used DES to evaluate the congestion in interconnected mines. The study found that specific places for debris removal are necessary to avoid delays in project completion. They showed that this technique is more practical compared to traditional methods and optimization models for project planning. The technique allowed for simulation of dispatch logic, evaluation of congestion at specific points, and understanding of maximal development capabilities. The results showed the importance of considering the associated variables in the plan to avoid significant delays in execution and subsequent start-up. Strauss et al. [34] created a rule-based DES to enhance the efficiency

of emergency medical services in Germany and Switzerland. Their model handles issues regarding ambulance location, dispatching, routing policies, and medical services collaboration with other prehospital care providers. The authors presented three common use cases from a practitioner’s viewpoint to demonstrate how the research outcomes are utilized to optimize specific areas in Germany and Switzerland. They showed that by using the rule-based DES, decision-makers can improve the effectiveness of regional emergency services without raising expenses.

Hong and Chien [35] developed a scheduling and dispatching system using a DES model to optimize the manufacturing system. The system considered bottleneck processes and dispatching rules, using a hybrid genetic algorithm to minimize photolithography capacity loss considering waiting time constraints. Optimal dispatching rules were determined for non-bottleneck workstations. The study was validated empirically and found to be practically viable. Lang et al. [36] discussed the use of RL to solve production scheduling problems in real time. However, there was a lack of information about the process of developing and implementing RL applications in related articles. The authors presented a method that modeled production scheduling problems as RL environments using DES and the Gym interface of OpenAI. The proposed method allowed for the deployment of pre-implemented RL algorithms and agents for both the allocation and sequencing of jobs. The authors also suggested that the method could be applied to other DES tools with minor adjustments. The authors planned further research to explore the extension of the method to non-Python-based DES tools.

Greasley and Edwards [37] conducted a literature review that explores the use of DES and big data analytics approaches, such as data mining, machine learning, data farming, visual analytics, and process mining. They highlight the differences between these two areas, with DES being a mature operational research tool that uses a graphical interface to create process models, while analytics approaches involve coding algorithms to analyze big data. Despite this divergence, the review identifies a small but growing number of use-cases and proposes a framework for a DES development methodology that incorporates data analytics techniques. The authors concluded that the integration of these techniques into DES has the potential to enhance the capability of DES in various stages of its methodology. Furthermore, DES can be used in a data farming role to drive data analytics techniques. To achieve these benefits, the authors suggested a research agenda to integrate data-driven and model-driven methods and encourage DES practitioners to expand their skills into the areas of data science.

3.2 Digital Twin-based applications in industry

The core principal of a DT is a digital equivalent for a physical entity of an asset. Simulation, including DES, is an important integral part of DTs. DTs cannot be considered as just simulation models. They stretch beyond traditional simulations with the aid of IoT technologies, especially real-time synchronization. Simulations are typically used for off-line optimizations in the design or planning phase. DTs go further to include the execution and variation in real-time. Not only do DTs help understand "what may happen" as simulation models, but they also provide understanding of "what is happening".

DTs offer an opportunity to monitor production systems in real-time and apply proper adjustments. The DT concept has been applied in many disciplines of production systems. Wanger et al. [23] introduced a DT architecture to monitor manufacturing processes and to control in real-time. Their system traced performance, analyzed real-time data, and simulated potential improvements. Zhang et al. [38] applied multi-objective optimization with the aid of a DT. Their methodology was applied for a hollow glass storage system. The optimization problem included storage allocation, vehicle scheduling, and outbound matching optimization. They used the synchronized real-time data for on-line adjustments. These adjustments were validated within the DT before implementation in the real system.

Vachálek et al. [15] focused on the continuous optimization of production processes through the application of the DT concept. For the digital part, they used Plant Simulation, a simulation tool made by Siemens. The process time data of the physical system was transferred to data storage, through which the digital equivalence was achieved. Production cost and time minimization was done using a genetic algorithm. They reduced the production time by 5.2% through scheduling only through off-line and on-line optimization. Cai et al. [39] introduced a virtual machine tool, as a DT, through the integration of both manufacturing and sensor data for a 3-axis milling machine. They intended to track the machine's behavior to diagnose/prognose failures and performance deviations.

Santos et al. [40] analyzed the applicability of DES as a DT in a non-automated process to implement I4.0 solutions. The authors proposed a method for conducting simulation projects in this context that integrates simulation with process data and updates it as per changes in the real environment. They verified its applicability in a use case for an aeronautical industry logistic process, where it is used to optimize decisions related to supply routes. The simulation model was integrated with the operation's enterprise resource planning. They created a decision-making interface for more efficient decision-making. The authors also discussed the use of simulation as DT, highlighting characteristics such as reaction time

and degree of autonomy. They proved the possibility of the proposed approach to reduce challenges by using versatile tools such as simulation. They suggested research opportunities related to creating and integrating several DTs in the same operation for broader decision-making through simulation.

Reed et al. [41] proposed a new modelling cycle for developing and analyzing DT models using DES technology, which differed significantly from the traditional modelling cycle for digital models. The proposed DT-specific cycle involved data queries, a model generator, validation routines, results analysis routines, and optimization implementation routines, allowing for automated optimization of the real system based on its current state in near real-time. The authors suggested that specialized software tools tailored to the unique characteristics of DT DES modelling were needed to facilitate the adoption of DT in the manufacturing industry. They highlighted the need for case studies demonstrating the applicability of the proposed modelling cycle in manufacturing industry applications and further work on mapping the DT modelling cycle to emerging standards and reference architectures. Qui et al. [42] improved the theoretical framework of the DT Workshop model by combining agent-based DES with reinforcement learning. They applied the evolutionary DT model to a real-world case and compared its performance with heuristic and random policies. Results showed that the model can self-evolve and adjust its operation strategy in real-time based on its current state. The proposed method effectively mitigates information and data fusion problems in complex multisource heterogeneous systems. The authors suggested using unsupervised learning techniques to determine relevant metrics and distributed integration methods for large-scale systems in future studies.

Mykoniatis and Harris [43] implemented a DT emulator of an automated mechatronic modular production system, which was linked with programmable logic controllers to exchange near real-time information with the physical system. They used a novel hybrid simulation and data-driven modeling approach combining DES and Agent-Based Modeling paradigms for the development and deployment of the emulator. The emulator provided support for design decisions, tested What-If scenarios, verified and validated the actual behavior of the complete system offline, tested realistic reactions, and provided statistics on system performance. The hybrid simulation approach helped overcoming limitations of standalone simulation languages and facilitated insight into the design of future production systems. The authors concluded that the emulator was a reasonable and proper solution due to its high fidelity to the physical system. It also prevented substantial installed system problems earlier in the design and development phases. Their future suggested works included exploring potential limitations of the communication and synchronization approach adopted in this work and analyzing more complex systems and behaviors.

Lugaresi and Matta [44] proposed a method to automatically discover manufacturing systems and generate digital twins with an appropriate level of detail for estimating system performance. Their approach retrieves relevant characteristics of a production system from data logs. It was tested on two cases and a real manufacturing line. The results proved its effectiveness in generating digital models that correctly estimates system performance. The proposed automated generation and tuning method can positively contribute to real-time simulation applications, with an updated and reasonably detailed model of the physical system. The authors acknowledged limitations to their proposed approach, such as complex identification of system structure in job shops and the need for careful selection of model tuning rules. They also noted that model accuracy may be limited by recorded events in the log, and further research is needed to determine minimal production observation requirements.

DTs are useful extension of simulation models. They have the capability to continuously follow their physical twin's variations. Thus, they do not suffer from being obsolete because of the real-time data synchronization. DTs provide updated simulation environment for the optimization or data analysis methods that provide corrective or proactive solutions. Yet, the quality of these solutions is limited to their algorithms' capabilities.

For traditional optimization algorithms, the solutions may not be the optimum for long times. In this case, new simulation runs are required to provide new optimum solutions. For DTs that are supported by ML or data analysis algorithms, the problem of long runs is solved. For such DTs, the algorithms should be well trained to handle different states of the real system. Otherwise, they will have to be retrained to improve their performance, which can also be time consuming.

3.3 Decision-making decentralization and holonic manufacturing

Decentralization and holonic manufacturing are both approaches to organizing production systems. Decentralization of decision-making in production systems is the process of delegating decision-making authority from a central authority to different levels of the organization. This approach is one of the key enablers of I4.0 [45]. It offers several advantages, including faster decision-making, increased flexibility, and better use of resources. Decentralization can lead to more efficient use of materials, equipment, and personnel, reducing waste and improving productivity [46].

Holonic manufacturing is an approach that is based on the concept of holons. Holons are autonomous, self-organizing units that can operate independently while still being part of a larger system. Holonic manufacturing is characterized by the use of autonomous, intelligent

agents, holons, that cooperate/collaborate to perform tasks and achieve certain objectives. This approach aims to improve flexibility, adaptability, and agility by creating a highly distributed and self-organizing production system [47].

Rossit et al. [48] proposed a data-driven scheduling architecture for manufacturing systems, which integrates physical and decision-making aspects of manufacturing processes. Their goal was to achieve decentralization and autonomy. The system has real-time access to data, enabling proactive scheduling decisions. The implementation of this architecture utilizes CPS technologies and vertical and horizontal integration. This achieved an embedded decision-support running on a data-driven engine utilizing big data techniques for information extraction.

Derigent et al. [49] conducted a literature review to identify the key enablers for I4.0. They assessed the extent to which holonic control architectures can fulfill these enablers. They presented a list key enablers. They described the holonic paradigm and highlighted how this architecture can partly fulfill the I4.0 enablers. They grouped the reference architectures into categories and discussed the research perspectives on the remaining unfulfilled key enablers. They concluded that investigating some research directions could lead to the development of sustainable, fully integrated, and evolutionary holonic architectures that fulfill the key enablers needed for I4.0. These directions include AI functionalities, DT-based architectures, and human-in-the-loop integration.

3.4 Reinforcement Learning applications in industry

Artificial Intelligence (AI) is recognized for many applications including computer vision, speech recognition, and other applications. Its training relies on showing desired input-output examples [50]. They learn from data to make proper decisions that increase their artificial reward [51].

ML has many theoretical capabilities to overcome common manufacturing challenges including [52]:

- Dealing with the complex nature of results and providing proper decisions. This is due to its ability to recognize patterns within the data and derive predictions for future behavior
- Adapting to dynamic environments. It learns and adapts to changes through its experience within the training environment
- Increasing existing knowledge by learning from results. It creates new information and

strategies [53]

- Identifying relevant correlations and/or causality [54]

ML algorithms have three types: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the algorithm learns from examples that are provided by an external supervisor. The data are labeled so true answers are used for supervising the algorithm [55]. In unsupervised learning, there is no supervising feedback such as in supervised learning. The algorithm is supposed to identify clusters from the data, which is unlabeled, based on similarities [56]. In Reinforcement Learning (RL), an agent is trained within an environment. It has to take proper actions that are expected to yield the best results. This is done by trial and error instead of being supervised or told. It emulates the process of how humans learn [23]. RL is most suitable when there is no knowledgeable supervisor. Thus, agents need to learn from interaction with their training environment.

A huge amount of data is normally provided in production systems upon adopting the concepts of digitalization. Along with increased process complexity and product variety, the operational decisions become difficult [57]. Although there is a significant potential from analyzing these data, only a small fraction is utilized [58]. Learning algorithms have made large progress in the past several years such as mastering the game of Go without human knowledge [25]. However, few serious applications are made in industry.

Adaptive management strategies have become important to tackle the challenges in complex production systems [59]. Heuristics are traditionally applied to solve optimization problems with a limited computation time and incompatible complexity. For dispatching, rule-based heuristics are widely applied in manufacturing systems. It is known that no rule outperforms all the others under any objective. Also, multiple rules may contradict each other [60]. Optimal dispatching depends on system's state. Hence, learning a strategy for the selection of dispatching rules is a promising approach [61]. In this context, RL has the following advantages [62]:

- Application with a limited scope
- Fast real-time decisions
- Cheap training
- Complex environments handling

Wang and Usher [63] applied RL to agent-based production scheduling. They used Q-learning to solve the dispatching rule selection problem. Their research was encouraged by the RL's

ability to provide autonomous decision-making for agents. They adopted the concept of decentralization of decision-making. It proved its importance in reaching the next generation of manufacturing systems with better reconfigurability and adaptability [64]. Their agent was able to learn to favor the best dispatching rule for each system objective. Stricker et al. [5] proposed a RL-based strategy for order dispatching in a semiconductor manufacturing system. The results were compared to the traditional rule-based system's results. As compared to the conventional heuristic approach, the proposed algorithm outperformed it. The algorithm achieved higher utilization and lower cycle time.

Waschneck et al. [22] applied the Google DeepMind's algorithm for RL to solve the dispatching problem. They trained several RL agents that apply DQN. They used a DES model to simulate the real system and to provide the training environment. The default scheduling rules, based on heuristics, were used to provide a state-action pairs for the DQN. This is considered to be a pre-training for the network before the RL training. The algorithm could reach the defined-objectives with less variance, compared to heuristic-based rules.

Kuhnle and Lanza [65] discussed the potential of using RL for production planning and control in CPS. They stated the potentials of decentralized decision-making, enabled by intelligent agents, to handle complex systems than rigid and static architectures. The authors showed that real-time data provided by CPS can pave the way for the application of data-driven algorithms to enhance operational efficiency. They implemented RL for order dispatching and maintenance management. For order dispatching, their approach outperformed the benchmark First-In-First-Out (FIFO)-heuristic. The application utilized a single agent that dispatch orders to equipment found in its production cell. The agent was penalized by a fixed reward value of zero upon selection of infeasible actions. The authors stated that further research is needed to broaden the scope of applications and design more robust RL algorithms.

Liu et al. [66] proposed a deep reinforcement learning approach to solve the job shop scheduling problem in dynamic environments. They use an actor-critic architecture with convolution and fully connected layers and a combination of simple dispatching rules as actions. The multi-agents were trained with a common neural network in a parallel training scheme. Their approach evaluation showed that it outperformed traditional dispatching rules and achieved a good balance between makespan and execution time. They suggested future work in adjusting the reward, transferring the deep reinforcement learning model to other scheduling problems, and accepting variable-sized inputs.

Lin et al. [67] proposed a smart manufacturing framework based on edge computing and use the DQN to solve job shop scheduling problem. Unlike the classical DQN that only makes

one decision, this work extended the DQN to consider the decisions of multiple edge devices. Through simulation results, their proposed method outperformed other methods using only one dispatching rule. Overall, this work highlighted the potential of using edge computing and RL instead of fixed heuristics to solve dispatching problems.

Altenmuller et al. [68] demonstrated how RL can be applied to the complex task of order dispatching in a job shop with strict time constraints. They used a Q-learning algorithm combined with a discrete-event simulation of a semiconductor wafer fab. Their study showed that RL can be successfully applied to manage order dispatching in a complex environment. Their results suggested that modeling rewards is a critical part of successful RL applications, particularly for constraint problems such as order-based time constraints. The authors noted that there is potential for more improved results by elaborating and optimizing reward functions. They also stated that human acceptance of autonomous decisions is one of the significant challenges to be addressed.

RL has been shown to outperform heuristics for solving dispatching problems in manufacturing systems. Unlike heuristics, RL agents adapt to changes in the system and learn from experience to improve decisions. This is particularly important in complex job shop scheduling problems, where the number of possible solutions can be enormous. DQN, which combine deep learning and RL, have been successfully applied to solve dispatching problems. These models have the ability to make decisions based on multiple inputs and can be adapted to include edge computing to reduce response time.

Through the literature review, some research gaps were identified. Specifically, there was a lack of a comprehensive approach that handles the complexities of real production systems, which often involve multiple agents that need to cooperate to achieve the overall system goal. One significant gap was the lack of implementations of simulated training environments that could account for real system level complexities. Most of the previous research in this area was based on simplified scenarios that did not accurately reflect real aspects of manufacturing systems. Further, the literature review revealed that some challenges related to the transition towards implementing AI in production system were not adequately addressed. For example, the human operators were not considered properly during the training of RL agents, despite their crucial role in the overall performance of the manufacturing system. In some system settings, human operators may result in applying different actions than those recommended by AI. Also, as a resource, they create variabilities due to their human bias. This gap needed to be addressed to ensure that the RL agents learn to operate in a real-world context and work effectively with human operators. In addition, the reward function formulations, found in the literature, do not cover special events of in which an event is resulted by the direct

actions of more than one agent. For instance, handling time constraints in more than one processing step such as timers in semiconductor industry. In such case, if assigned rewards for each agent's action are not related to the state and the other agents' actions, the learning becomes slow and yields trivial actions.

In RL applications for production system, the literature does not provide a specific approach to address the impact of rare events, such as resource interruptions, not failures, during the training of agents. These events may not receive enough consideration during training due to their infrequency compared to other events. Since training time is limited, a strategy is needed to encourage agents to account for these rare events during their training.

Accordingly, the thesis proposes an original approach that uses multiple agents with decentralized decision-making based on the concepts of holonic manufacturing and decision-making decentralization. The agents learn independently using DQN and apply decision-making decentralization, which allows for faster decision-making and increased flexibility. Inspired by holonic manufacturing, which focuses on independent units working together, and decision-making decentralization, which spreads decision-making authority, the proposed approach was developed.

The approach also formulates state information and reward functions that support cooperative behavior among agents and considers real system variabilities. The thesis introduces a self-play scheme for training managing agents for handling less frequent events, mainly system resource interruptions, and employs them to improve the learned policies without prolonging the training period. This approach involves conflicting agents that apply the concept of self-play in a production system context. Similarly, the thesis proposes a self-play scheme between the system and customers that vary their demand rate, which trains the managing agents for handling variable customer demands in an adaptive way according to their learned policies.

The proposed approach offers significant contributions to complex systems, specifically the field of semiconductor manufacturing, by providing a comprehensive solution to handling the complexities of real production systems. Overall, the thesis presents an original approach that can help advance the field of semiconductor manufacturing and has implications for other manufacturing domains as well.

**CHAPTER 4 ARTICLE 1: A DISCRETE EVENT SIMULATION LOGIC
FOR SEMICONDUCTOR PRODUCTION PLANNING AND CONTROL
WITHIN INDUSTRY 4.0 PARADIGM**

Ahmed H. Sakr, Soumaya Yacout, Samuel Bassetto

Published in:

*Proceedings of the International Conference on Industrial Engineering and Operations
Management Toronto, Canada, October 23-25, 2019*

Published online in:

December 20, 2019

4.1 Abstract

The recent advancement in digital technologies has encouraged manufacturers to adopt intelligent data analysis and decision-making support tools to leverage their competitiveness. They have become challenged by the need for more flexible and complex manufacturing systems. This complexity and flexibility cannot be handled efficiently with traditional production management paradigms, such as lean management. Semiconductor industry is an example of a complex production, in which high level of flexibility is required to meet a continuously changing customer demand. This limits the efficiency of static modeling approaches and it reflects negatively on the overall production yield.

In this research, we propose a discrete event simulation logic for assessment of several challenging production management-related problems at a semiconductor manufacturer. The logic includes scheduling and processing a large variety of products, while respecting relative priorities and customers' needs. It also accounts for production resources, equipment and operators, with variable availability. Our simulation logic is data-driven, in which data that drives the different actions taken within the execution of the simulation, is compiled. Consequently, the model is highly flexible to continuous changes in the model's parameters. The proposed logic provides a quick and risk-free tool for capacity planning and reliable experimentation with different 'What-If' scenarios.

Keywords

Complex production, Discrete event simulation, Data-driven Agent-based model, decision-making support, Industry 4.0.

4.2 Introduction

The recent advancement in digital technologies, which is adopted by Industry 4.0 paradigm, have put a challenge for traditional production management-based industry, especially if the product or production is normally complex. These communication and information technologies, based on the Internet of Things (IoT) perception, help interconnecting shopfloor components for better management and control [2]. Moreover, the IoT allows the exchange and collection of information between different system components, things, through the internet. This gives a chance for smart integrated systems that communicate with each other to achieve a certain task or to improve the general performance of the system. Similarly, the same concept is industrially applied which gives opportunity to monitor the system compo-

nents, devices or equipment, remotely. This allows the integration between physical systems and virtual models which results in more representative and reliable decision-making support tools, hence more efficient and profitable production strategies [2, 3, 13].

There are different approaches to virtually model that represent a physical system. These approaches differ in whether the system is static or dynamic, and whether it is deterministic or stochastic. According to the structure and the dynamics of the system, the suitable type of model should be selected in order to provide a good level of similarity and valid simulation results. The model parameters should be updated regularly for more accurate reality representation. This becomes more crucial in case of complex production systems where such parameters are continuously changing according to different states of the system. A product or production is considered to be complex if it is subject to variable customer demands, or if its production flow is not fixed [9]. These parameters may include, but not limited to, the different restrictions or qualifications of the production resources. In addition, other managerial decisions that may affect different products' relative importance. Upon respecting such modeling aspects, the model will be able to accurately mirror the reality and provide consistent information for decision-making support. Moreover, this will provide a reliable tool for conducting many experiments and answering different 'What-if' scenarios [15].

In semiconductor industry, the fabrication areas are currently considered the most complex production systems. Traditionally the focus was on the design consideration and yield improvement rather than on the production management. However, the manufacturers have been always seeking for some tools that put the focus on the production management as a key solution [18]. This industry is defined as high-mix of products at low-volume with an end-user market-based demand which adds a source of complication for the production [69]. The discrete event simulation is considered a successful tool for performance assessment/improvement and decision support in this industry [18]. The success of discrete event simulation lies in its ability to better represent the dynamics of the system along with its stochastic nature. The simulation for this complex production should consider also the dynamics of system parameters that the model is built on. This is a new consideration for simulation of dynamic systems, and the foundation for model-based decision-making support within the Industry 4.0 concepts [15].

In this paper, we present a simulation logic for semiconductor manufacturing system modeling with a discrete-event simulation. The proposed logic is designed to be applied using Arena® simulation software. We address the different aspects that should be considered in modeling such system, and how to include it to the model's logic efficiently. The proposed logic includes

the structuring of the input data that is changing continuously. It is designed to be ready for accepting different variations in the system parameters with no need for redesign. Our simulation logic is data-driven, but it also includes agent-based logic that controls the flow of products between the different model's subsystems and modules.

4.3 Problem statement

4.3.1 Semiconductor industry background

The manufacturing process in semiconductor industry consists of a repeated cycle of several processes, which are applied to the wafers. A sequence of cycles is performed. In each cycle, a layer is built on the wafer through different processes. The number of cycles depends on the complexity of the produced device. Within the processes, wafers are tested for quality conformance before proceeding to the next steps. The wafers are either bonded with the same or different wafers, or diced into pieces that are called dies, according to the customer requirements [18]. The manufacturing firm usually consists of several fabrication areas with clean rooms environment. Each clean room contains several work areas that consist of several equipment of the same type and capabilities.

4.3.2 Process description and sources of complexity

The wafers are moving in lots from one step to the other. A lot is a cassette that has a fixed number of wafers of a certain size. The wafers are processed by the equipment according to its capacity. For example, some equipment handle only one wafer at a time and requires manual feeding by the operator, while others can be loaded with the whole cassette, or even more than one cassette. Other equipment process part of the cassette, hence it requires splitting the lot into unit batches according to the equipment capacity. Some equipment are clustered in one single equipment that normally has an inside robot manipulator to handle the wafers. Accordingly, it processes several steps without the need for an operator for loading and unloading or transferring the lot. Normally, the operators are qualified to operate many equipment types according to the level of training they achieved, but not necessarily all equipment.

The processes that are achieved in each step are called recipes. Different products that have the same recipe can be processed in the same equipment, simultaneously. However, some recipes may be restricted from being processed by certain equipment due to quality issues. These restrictions are defined according to both product and recipe. Some recipes have a constraint of minimum delay between consecutive steps. In this case, the wafers have a

waiting time before being processed in their next step. Other recipes have also a maximum delay constraint, or a timer limit, before which the wafers should reach a specific step. If the limit is exceeded due to different delays, the lot is scrapped in case of no possibility for a rework. The part that needs rework is sent back, while the rest of the lot proceeds to the next steps until a certain rejoin point, where it waits for the reworked part. According to the process flow of each product, some steps contains if-conditions in which the lot is split into two parts according to the result of the condition. These parts are rejoined at a certain step. Usually, the fabrication areas are not connected to each other. Manual transfer is thus required. Specific operators are only responsible for the transfer of lots between different fabrication areas.

There are different aspects for assigning relative importance, priority, to each lot. These aspects may include: the customer or product importance, the slack time, and most importantly how close to the timer limit in case there is an active timer for the lot. These priorities are used to order the waiting lots in each equipment station so that the lot which has the highest priority is processed first. The priority that is related to timers is continuously changing after each step to reduce the number of junked wafers due to the exceeding the timers' limits.

As a result of all these aspects that should be considered during processing the lots in its different steps, the process flow becomes highly dynamic. Consequently, the flow of the product is no longer fixed and is subject to stochastic alternatives that depend on each of the product itself, the manufacturing resources availability and qualification, the managerial decisions, and the variable customers' requirements. The decision-making tasks become very hard when such system is not properly modeled, hence it is nearly impossible to assess the impact of any taken decision.

4.4 Methodology

4.4.1 Selecting the appropriate simulation modeling type

Due to the high level of complexity in such manufacturing system, it is clear that such system cannot be handled efficiently with traditional production management paradigms such as lean production [70]. Such system is highly dynamic as it is vastly time dependent. It also contains many probabilistic events such as the uptime of the equipment. As shown in Fig. 4.1, there are four types of simulation models. The Discrete-event simulation is the most appropriate for semiconductor industry modeling due to its dynamic probabilistic characteristics.

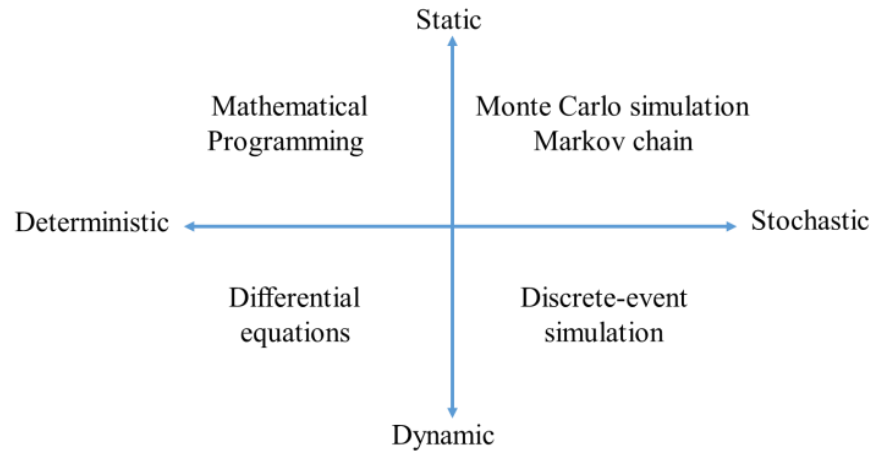


Figure 4.1 Different types of system modelling

4.4.2 Data-driven agent-based modeling

In semiconductor industry, the products are processed according to their process flow, which consists of several steps that include specific recipes. These recipes are defined and shared among different products. Each recipe has information regarding the type of equipment that is needed, the processing time, and the information that specifies the minimum and maximum delay among steps, if any. The process flow of each product is not fixed. Some steps are added or deleted due to the possible production technological improvement. For example, some production and inspection steps are possible to be performed by a single new equipment. In this case, the process flow must be updated to account for this upgrade.

Also, the equipment restrictions for recipes and/or product depend on some aspects including the equipment state which are continuously varying. For all these reasons, we adopted a data-driven modeling technique rather than product-driven. In data-driven models, the data controls the product flow among different steps in the model. This gives the flexibility to adopt any possible variation in flow, delay or restrictions, without rebuilding the model. It also has the advantage of building relatively more compact and simpler model to represent complex production systems that normally contain large number of resources. A simple model means that any possible errors are tracked and fixed, and any desired logic modification are easily applied.

The agent-based models have the advantage of using agents that check various aspects in the model. Accordingly, they take predefined actions that are applied to the lots to move them through the model. Arena has the capability to carry out agent-based simulation by using logical entities that go through a series of blocks or modules to execute specific functions

within the model. These agents can perform actions such as moving products represented by other entities to different locations or removing them based on certain simulation states.

The advantage of having agents within the model is that many human logical procedures and actions are modeled by them, in addition to the flow of products themselves. In our logic, agents simulate the operators' logics, with different schedules and interruptions during their work shift, and the execution of complex logic such as processing a lot within a timer's limits. They are also data-driven, which means that their actions are defined according to the input data. These actions are ready for any alternation to reflect any variation in the actual system.

Our model logic has the advantage of combining the complexity of an agent-based model, and the simplicity of a data-driven model. Also, it has the ability of continuously reflecting the actual system's state without the need for rebuilding or modifying the model itself, because the data will do this task. This is in accordance with the new paradigm of simulation modeling that is among the key technologies of Industry 4.0-based manufacturing.

4.4.3 Input data structuring

The data is structured and is fed to Arena using MS Excel. This MS Excel is updated by the actual system databases to continuously providing the most updated system parameters for the simulation. The input data consist of the following:

1. Products basic data
2. Process flow data for all products
3. Recipe information
4. IF-conditions information
5. Rework information
6. Timers
7. Equipment information
8. Equipment restrictions
9. Operators information

The product basic data contains an ID, which is given for each customer and product to be used for assignments of different information within the model. It contains the interarrival time distribution function or schedules for each lot production at start, and the distribution of the lot types for each product. The lot type refers to whether the lot issued to satisfy a customer request, or it is for testing and process development reasons. A default priority is given for each customer, product, and lot type. This priority is usually predetermined according to the relative importance of each customer or products. The process flow data includes all the processing steps for each product, with all important information such as: the recipe name and ID, the fabrication area, the type of the step whether it is a normal step or an if condition step, the condition parameters to be checked, the possible rework the rework and rejoin steps for the lot splits, the then/else steps of an if condition, and the alternative recipe if any.

The recipes are indexed with an ID to be easily accessible by Arena. Each recipe has the information about its fabrication area, equipment type, the minimum delay and timer if it requires a time limit, the average yield based on calculations from the historical lots data, the recipe processing time and operator time, the recipe type whether it is normal, bonding, dicing step, and if this recipe has an equipment restriction.

The ‘if’ conditions are defined for each product and the average percentage of having a ‘Then’ statement is defined based on the historical lots’ data. Similarly, the rework information for each product at each step is defined.

The timers are defined for each process flow of each product. This includes an ID for that timer to be accessed by Arena, the time limit of that timer, the number of steps within the timer, and the start and end steps. This information is used by a special logic for processing lots that have a time limit for delay.

The equipment information includes an ID for each equipment, along with an ID for its type and its fabrication area. The capacity is also defined with the number of wafers or full cassettes that equipment processes. Also, the uptime/downtime distribution for each equipment, based on the history of that equipment, is included. These distributions are used by the agents that are responsible for simulating the availability and the failure of the equipment. The equipment restrictions’ information contains the restrictions or dedication of specific equipment for certain recipes, or certain recipes for specific products. This is used when assigning an equipment to process the lot at each step.

The operators’ information includes the operators’ IDs of each fabrication area and their qualification for different equipment types. It also includes their working shifts and break times to be used for their activity simulation by the agents who are responsible for this task.

Finally, it contains the interarrival times for interrupting events and their time distributions for simulating possible interruptions of the operators during their working shifts.

4.4.4 Simulation logic

The input data is ingested to the model to drive its different actions. The run logic begins with reading the input data and assigning it to appropriate variables or expressions that can be handled by the simulation modules. Then different agents are created to execute different tasks according to information from the input data; specifically, the number of operators, the number of the equipment types, and the total number of equipment.

Figure 4.2 shows the working logic for the equipment stations' agents. These agents are responsible for moving the lots from the queue of each equipment type to be processed. This logic helps respecting the priority for the waiting lots, and provides simplicity for the modeling as each lot may have restrictions or dedications for different equipment within the same equipment type station, hence the traditional method of seizing resources, equipment and operators, will be more complicated to realize the same logic. The agent searches for all other waiting lots that have the same recipe to be processed with the first waiting lot according to the equipment capacity, as usually the equipment cannot be accessed once processing a lot.

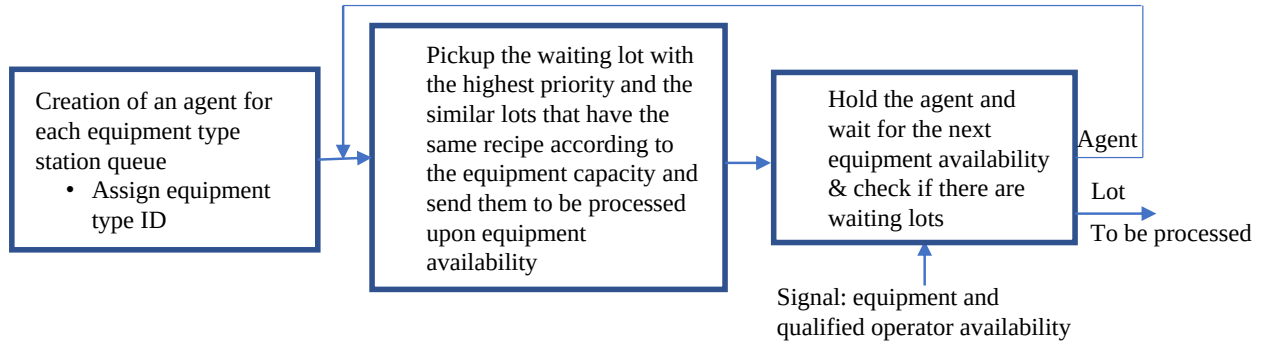


Figure 4.2 Working logic of equipment stations' agents

This logic is facilitated by the attributes, or information, that are assigned to each lot including its product ID and step ID. The agent identifies the lots through this information. Accordingly, it acquires other information regarding the lot's current step from the input data. This also reduces the unnecessary duplication of data and eases its synchronization with the actual system database for continuous updating.

Figure 4.3 shows the working logic for equipment uptime/downtime agents. The correspond-

ing agents are responsible for assigning active or failure states to their equipment according to the time distribution functions for both uptime and downtime. The distributions are obtained, from the input data, during the model run. These distribution functions are calculated from the history of each equipment, and these calculations are updated regularly, to accurately detect any change in the uptime/downtime due to improved maintenance policies, for example.

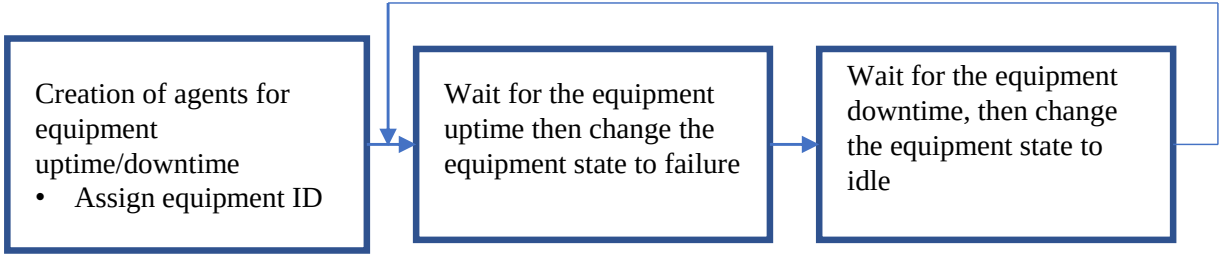


Figure 4.3 Working logic of equipment uptime/downtime agents

Figure 4.4 shows the working logic for the operators' schedules and interruption events' agents. These agents are responsible for assigning the active/inactive states to their operators according to each operator's schedule and assigning the active/failure states due to any possible interruption event. These events arrive randomly according to a given interarrival time and for a given duration which is calculated from records of observed interruptions for the operators in the actual system. The schedules are given in the input data for each operator, hence it is updated upon any possible modifications of the actual operators' schedules.

Figure 4.5 shows how the lots are created and how they are prepared for their first processing step. The important information is assigned for each lot upon creation. This information includes the product ID that is used to define the process flow and information about the steps. The relative importance information of a product is checked along with other prioritization aspects such as timer's limits. Consequently, the priority is assigned and is used for ranking the lot among others. The lots then wait, in the equipment station queue corresponding to their steps, for the equipment station's agent to move them for processing upon availability of the equipment and the qualified operator.

Figure 4.6 shows how the lot is getting ready to be processed in its current step. The lot is first checked for any equipment restrictions, then it is assigned to the available equipment and operator to be processed. The lot is then delayed for the processing time, and then it is unloaded by the operator from the equipment and is sent to the after-processing logic. If there is an equipment restriction, the lot is sent back to wait in the station's queue until

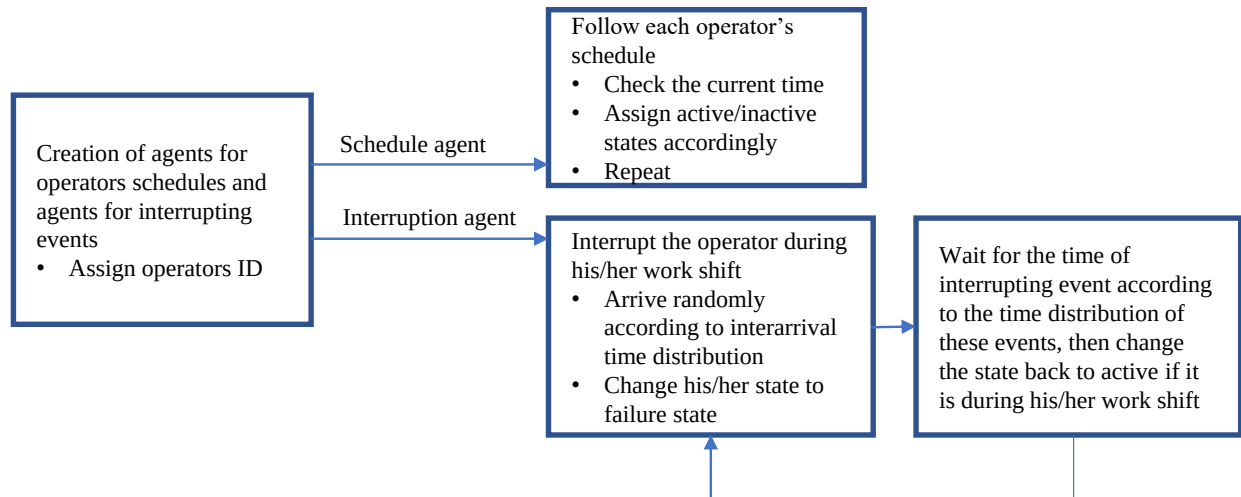


Figure 4.4 Working logic of operators' activity/interruption agents

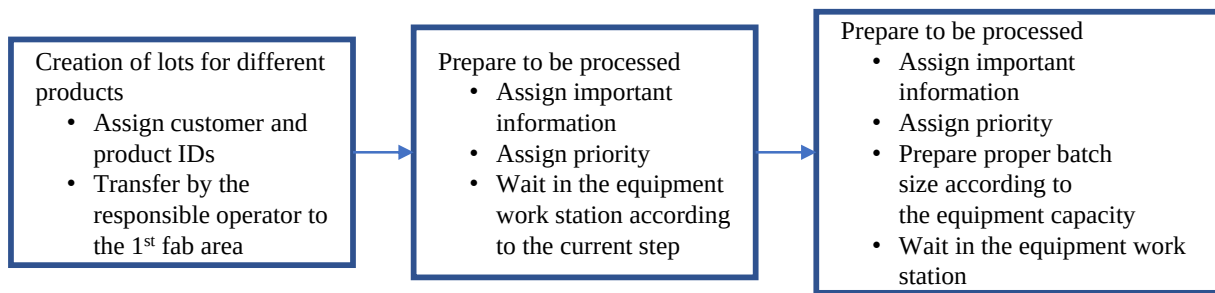


Figure 4.5 Lot creation and assignment of important information logic

another equipment becomes available within the same station.

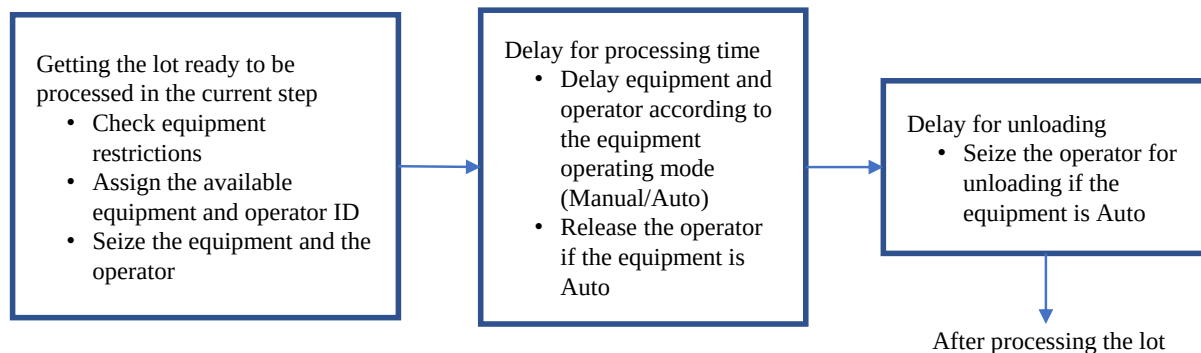


Figure 4.6 Lot processing logic

Figure 4.7 shows how the after-processing logic. This logic includes the following:

1. Batching the wafers from the same lot if the lot was not processed at once due to the limited equipment capacity.
2. Disposing the disqualified wafers from the lot according to the yield of each recipe.
3. Checking if the current step may have a rework procedure and if any wafers requires a rework. In case a rework is required, the lot is split and the part that requires a rework is sent to the corresponding rework step.
4. Checking if the current step is a rejoin step for a rework procedure. The lot will wait in this step until its reworked parts rejoin it.
5. Checking if the next step is an if condition. If yes, the lot is split and sent to the Then/Else steps of that condition.
6. Sending the lot to its next step. The lot may wait if the next step is not within the same fabrication area. In this case, the lot waits for the responsible operator to make the transfer.

The lots are subject to many splits such as reworks or ‘if’ conditions. A log is created for each lot within the logic and is realized by global variables in Arena. These logs are used for tracking of the splits of each lot according to the lots ID and the number of wafers in each split. Normally, the splits are subject for scrapping due to many reasons such as low yield recipes. In this case, the logs are important for rejoin steps to determine the number of lot’s parts that are rejoined to form a complete lot before proceeding to the next steps.

Due to the importance of processing of lots within timers, an agent is created for each lot that is entering several steps within a timer. This agent assesses the criticality of this timer by comparing the total processing time for the steps within the timer to the time limit of the timer. If the timer is critical, the agent seize all equipment in advance before processing the original lot within that timer, while respecting the priorities of the waiting lots. If the timer is not critical, the agent check if all required equipment are up or not, and decide to proceed or hold the lot. If the equipment is down, the agent waits until the equipment become up and then check if there is a waiting lot that is higher in priority than its lot. If there is no higher priority waiting lot, the agent’s original lot is processed. This logic respects the lots priority and processes the lots within timers safely to reduce the junked or reworked lots due to exceeding the timers’ limits.

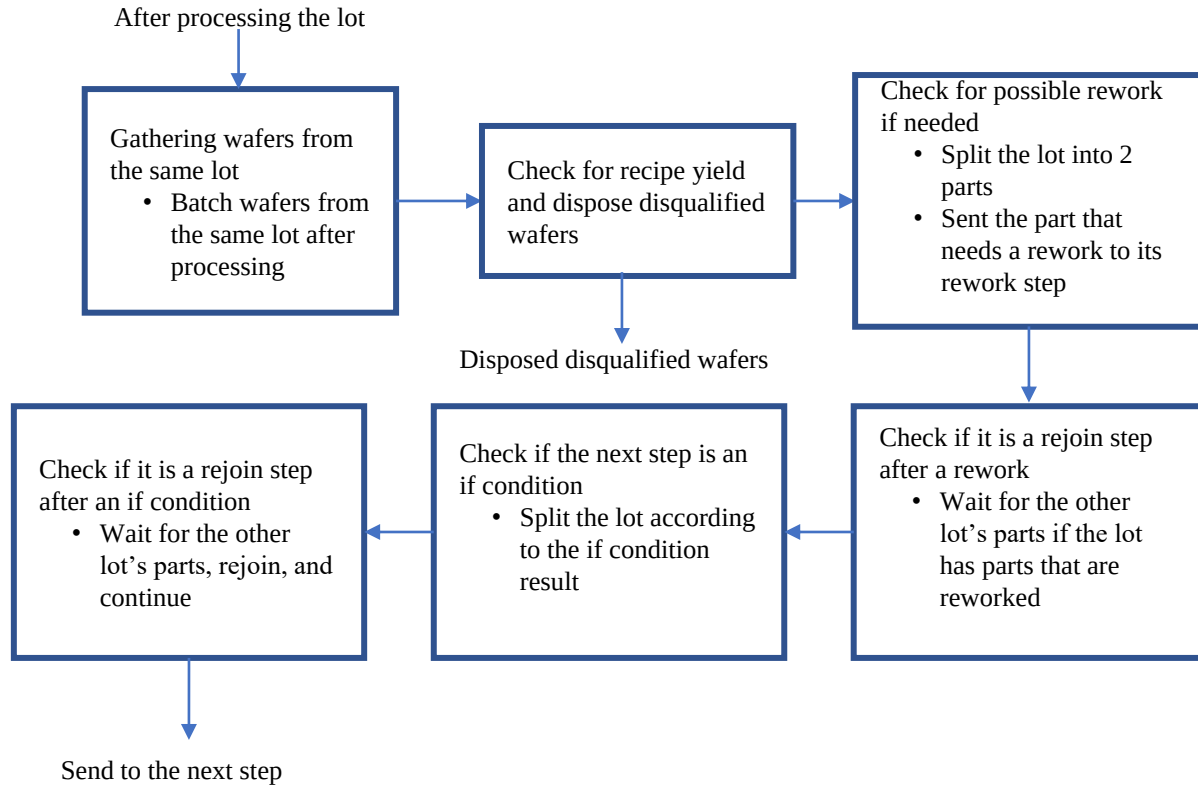


Figure 4.7 The after-processing logic

4.5 Goals

By applying our proposed logic, the simulation provides default results including the following:

1. Lots:
 - Total number in/out and total time
 - Value added/ non-value-added time
 - Waiting time
 - Total work in process (WIP)
2. Resources: the instantaneous utilization
3. Queues: average number and time of waiting in each queue

These indicators provide a general insight into different bottlenecks and their possible reasons. More details are acquired by using counters or tallies for each type of product to detect where and why the product has the most delay.

When addressing different problems in the semiconductor's industry, some manufacturers cannot assess the future impacts of their decisions due to the lack of a real representative model. Furthermore, problems are addressed in a reactive mode rather than a proactive. As a result, the cycle time increases dramatically for most of the products because of the increased downtimes or improper capacity planning. Upon realizing a representative model, decision makers are able to evaluate their different decisions and realize the possible upcoming problems in advance, and how to prepare the actual system to tackle them.

Our model logic is a basis for conducting different experiments and assessing different 'What-if' scenarios at no risk or any financial loss. This efficiency is due to the continuous synchronization of different parameters with the actual systems, which ensure that the model is always reflecting the reality. ANOVA is beneficial for evaluating the significance of different parameters on the various performance indicators of the model. This helps the decision makers to focus on solving different problems with the most significant aspects rather than applying temporary solutions without assessing their future impacts.

4.6 Verification and validation

After building the simulation model, it must be verified and validated according to its purposes. Verification is achieved if the modeling assumptions are correctly translated into a computer model. Normally, this is done by debugging the simulation model to detect any deviation from the modeling assumptions. For example, the model is to be tested by checking whether the priorities of the waiting lots are correctly assigned and respected or not throughout the model's execution. Thus, simulation models are verified to follow the modeling assumptions if no assumption is violated due to a modeling error. For validation, the model is to be tested for its representation accuracy of the actual system that it simulates. This is done within the scope of the particular objectives of the simulation study [71].

A good technique for verifying our simulation logic is to debug the model in modules or submodels. This is due to the various logical aspects that are included in the simulation such as assigning priorities, batching wafers, lot split and rejoin. This task is possible during the model building. Simulation model building is divided into building submodels in which each realizes a certain logic. Upon building a submodel, it is tested for its conformance to the relevant modeling assumptions. Hence, debugging and correction, if needed. The submodels are added and debugged successively until reaching the full model.

Simulation models validity is established when its outputs closely resemble the actual system's ones. This validity test is achievable through the comparison with the actual system

under the same conditions. After verification, historical data from the actual system are used to recreate the same manufacturing scenarios through the simulation model to be validated. This approach is called the ‘correlated inspection’ which results in comparable output statistics from both the model and the actual system. If the model’s outputs compare closely to that of the actual system, then it is considered a valid model. According to the correlation between the simulated and the actual outputs, the confidence in the model is built. Such comparisons are realized through numerical statistics. Simulation model validation relies on sample means comparison for many production scenarios. If the difference is negligible, then we conclude the model’s validity.

4.7 Conclusion and future works

The semiconductor industry is an example of a complex production due to the manufacturing technologies and the flexibility required to satisfy customer requirements. The manufacturers always seek for reaching production performance improvement in order to keep a high level of competitiveness. This implies the implementation of untraditional production management strategies that can handle the increased level of production complexity. Such problems are addressed by the applications of the Industry 4.0 concepts that adopt the IoT technologies and synchronized system models which copies the actual system continuously.

The basis of having a good representative model is selecting proper methodology and type of modeling that is able to simulate the behaviors of the actual system. Nevertheless, it should be continuously monitoring the actual system to adapt for any modifications that may occur. We proposed a methodology for modeling a production system for semiconductors using discrete-event simulation model that is both Data-driven and Agent-based. The proposed logic is easily applicable using Arena simulation software. Our logic included the flexibility for accepting any updates in the model parameters with no need for rebuilding the model. This was realized while keeping the modeling as simple as possible to be easily assessed and validated.

Future work for this research involves conducting design of experiments with the model results for real input data to understand different relationships between model outputs and different model parameters and inputs. This would help solving different problems and improving the general production performance of the actual system.

**CHAPTER 5 ARTICLE 2: BUILDING DISCRETE-EVENT SIMULATION
FOR DIGITAL TWIN APPLICATIONS IN PRODUCTION SYSTEMS**

Ahmed H. Sakr, Ayman Aboelhassan, Soumaya Yacout, Samuel Bassetto

Published in:

*Proceedings of 26th IEEE International Conference on Emerging Technologies and
Factory Automation, ETFA, Västerås, Sweden, September 7-10, 2021*

Published online in:

November 30, 2021

5.1 Abstract

Digital equivalence is the main objective of Digital Twin (DT)s, and simulation is an integral part. DTs reach beyond traditional simulation with the help of real-time synchronization through Internet of Things (IoT) technologies. Simulation supports off-line experimentations and planning, while DTs offer synchronous execution and modification. DTs help to understand "what may happen". Also, they present "what is happening" and its management methodologies. In this paper, we present building aspects and an integration approach for a Digital Twin based Discrete-Event Simulation (DES) model. Our approach utilizes a hybrid simulation involving a data-driven DES and agent-based simulation within a DT framework. It presents an integration layer that provides two essential features: reconfiguration and state initialization. It gives simulation models configurability and integrity that are required for operating within a DT. Our proposed approach is presented through a use case of a semiconductor manufacturing system. The proposed integration layer extends the usability of current DES for a DT within a Cyber-Physical System.

Keywords

Digital Twin, Discrete-event simulation, Industry 4.0

5.2 Introduction

Industry 4.0 is considered a digital revolution due to the Internet of Things (IoT) development. When compared to previous industrial revolutions, Industry 4.0 expands noticeably in the manufacturing sector. The changes made by this revolution designate future transformation of current production systems and their management schemes [11]. The basic concepts and technologies of Industry 4.0 include the use of IoT and intelligent processes in a Cyber-Physical System (CPS).

CPS provides tools for rapid reaction to variabilities and for decision-making seeking productivity improvement. It operates at both virtual and physical levels. It monitors, senses, and controls physical devices.

CPSs embed intelligence that is provided by employed big data models [3, 13]. Such intelligence is essential for data-to-information conversion. It includes several methodologies such as diagnosis, prognosis, and health management algorithms. Networking elements of CPSs allow real-time monitoring and synchronization of the real world's events with the virtual world of the Internet [3].

Within a CPS, a Digital Twin (DT) digitalizes its real system, extracts information/insights, and supports decision-making tasks [72]. Therefore, traditional production processes are complemented, optimized, and controlled through a digital world. A DT includes a Digital Shadow (DS) which acquires real system's state and creates a real-time digital replica. The replica is employed through its operating DT for simulation-based analyses and real-time management and control [73].

For a DS, it is important to replicate real system's state changes including configurational ones. Real systems evolve over time due to variability, reconfigurations, extensions, and upgrades. Hence, an operating DS must evolve to keep its level of abstraction and avoid misleading insights.

On production system level, Discrete-Event Simulation (DES) is a suitable tool for performance assessment and decision support [18]. It is employed in many contexts such as experimenting, assessing future actions, and real-time monitoring [74]. A DES model becomes a functional DS when real-time data utilization is applied. Such data are normally used to reconfigure the corresponding model and/or modify its state to have a similar image of the reality. A state of a real system includes information about the Work-In-Process (WIP) and system resources. This information is essential when a model is required to run starting from a specific point of time. In this case, the model should be initialized with the corresponding WIP rather than starting empty. Similarly, system's resources state are initialized, such as operators and equipment availability.

DES requires some core features to qualify to work as a DS within a DT. These features are also important for an operating DT to become functional within a CPS. The information of real system's components is needed to be available in a structured form including resources, processes, products, and management. In this paper, we address some DES model building aspects and required features within the DT-based simulation context. Our proposed approach, applied to a semiconductor manufacturing system use case, consists of:

- Structuring model's database
- Building a DES model using Arena simulation software
- Building data preprocessing and integration scripts to support model building and state initialization

The rest of this paper is organized as follows: A background for DES and DT are introduced in Section 5.3. DT functional requirements of simulation are discussed in Section 5.4. The

traditional DES modeling limitations are presented in Section 5.5. The proposed approach is presented in Section 5.6. Finally, the conclusion is provided in Section 5.8.

5.3 Background

Industry 4.0 is a paradigm that enforces digital transformation for the manufacturing sector. It aims at creating a digital network of physical assets through information and communication technologies [12]. The CPS's networking is one of the key technologies of the Industry 4.0. It facilitates real-time monitoring and synchronization of the real world's events to the virtual world of internet [3]. The main enabling technology in CPSs is the Digital Twin. Within CPSs, DTs enable simulation to optimize and control production lines continuously, based on real context [16].

5.3.1 DES of Production Systems

DES creates a dynamic stochastic simulation models that evolve in a time-dependent manner. In this type of simulation, the real system is represented by a set of modules. Entities, such as products, move through the model. Resources provide services to entities. System's state information is derived from attributes of each entity, or global variables. The state is updated upon advancing in each time step [19]. DES models have several modeling techniques. They include process-driven, data-driven, and agent-based.

Data-driven simulation is a technique where simulation models are fully parameterized through a series of data inputs, allowing users to create and execute simulation models without the need for programming/reprogramming. Essentially, the simulation is based entirely on the input data provided to the model [75]. It also has the advantage of building relatively compact models to represent large and/or complex production systems.

Agent-based models have agents that acquire model's state and take actions accordingly. The term "agent" describes a software component that perceives, acts, and communicates with other agents and its environment to complete certain tasks [20]. They are computational abstractions encapsulating control along with a criterion to control a task or a goal [76]. These models are required for providing different properties such as autonomy and responsiveness. Agents work according to a predefined logic. However, they can be designed to provide autonomy to adapt to changes and exhibit intelligence. In this case, they react to uncertainty and lack of information [77].

5.3.2 The DT Technology in Industry 4.0

The core principal of a DT is digital equivalence of physical assets. Hence, simulation is an integral part. DTs require adaptable simulation models. They reach beyond DES with the aid of IoT technologies, especially for real-time synchronization [78]. DES is typically used for off-line optimization in design or planning phases. DTs go further to include synchronous execution and modification. Not only DTs help understanding "what may happen" as DES models, but they provide understanding for "what is happening" and for reaction.

Communication with the real system is an essential feature in DTs. This requires an integrating layer for handling different conventions between the real system and its virtual counterpart [79]. However, the details of such integration is not properly addressed in the literature [80].

Negri et al. proposed scheduling optimization and health management through a synchronized DES model within a DT [81]. This includes model initialization through parameters update. Santos et al. programmed a DES model to periodically acquire some information from the real system so that the model reflects the real state [82]. Accordingly, the solutions that are offered by the model are updated. Murphy et al. used a Excel VB to prepare the input data which are read by the DES model [79]. However, the data structure and macros details are not elaborated. These studies noted the importance of an intermediate translating layer between the real system and the model.

5.3.3 Role of DESs in DTs

The main objective of DTs is to acquire real system's state and to provide adjustments or to support decision-making. This is generally required for tasks such as dispatching, operations and maintenance scheduling, and all other production management-related decisions. Decision-making tasks are achieved through set of heuristics rules, metaheuristic algorithms, or Machine Learning (ML) algorithms.

DESs are essential but not sufficient for operating within a DT without suitable synchronization with the real system. Flores-García et al. noted the importance of the linkage between real system's data and the DES model [83].

In this context, a DT evaluates its decisions through the simulation model. For example, heuristic rules are evaluated and ML algorithms are trained through experimental runs of the simulation model. Similarly, off-line optimizations, such as for maintenance policies, are also executed through the DT.

This paper aims to present building aspects and an integration approach for a DT-based

simulation. The proposed approach utilizes a hybrid simulation involving a data-driven DES and agent-based simulation within a DT framework. This provides simulation models configurability and integrity that are required for operating within a DT. Thus, an integration layer is introduced to provide reconfiguration and state initialization to extend the usability of current DES within DTs. The model building is achieved through a use case of a semiconductor manufacturing system to demonstrate details that are related to real systems.

5.4 DT Requirements

DT operates in both on-line and off-line modes. The on-line mode provides real-time decisions such as dispatching and resources allocation. The off-line mode provides decisions such as maintenance scheduling. Fig. 5.1 presents a functional DT. Its DS acquires physical system's configuration and state.

A DT employs a DT-manager which coordinates among all DT's components. For a real production system, a DT acquires its real-time virtual state through its included DS. Upon acquiring the state, the DT uses its tools to provide decisions/actions. These tools consist of, but not limited to, DES, optimization algorithms, ML and/or Reinforcement Learning (RL) algorithms. The manager trains/retrains learning algorithms off-line through the DES model. It loads the model information from its operating DS.

A DES model requires some features to be used within a DT. These features can be summarized as follows:

- Data-driven model execution
- Data-driven model configuration
- Agent-based decisions execution within the model
- Capacity for initialization from a certain state

The data-driven features are essential in any DS or DT as systems generally evolve over time. In this case, hard-coded models are not practical for such application as they need manual reconfiguration. For a fully data-driven model, the DS considers all operational and configurational changes that might take place. Such changes are captured through the system's database. This implies that the database must be structured in a descriptive form as discussed in Section 5.6.3. In DES, agent-based models allow execution of more complex logic, such as RL policies.

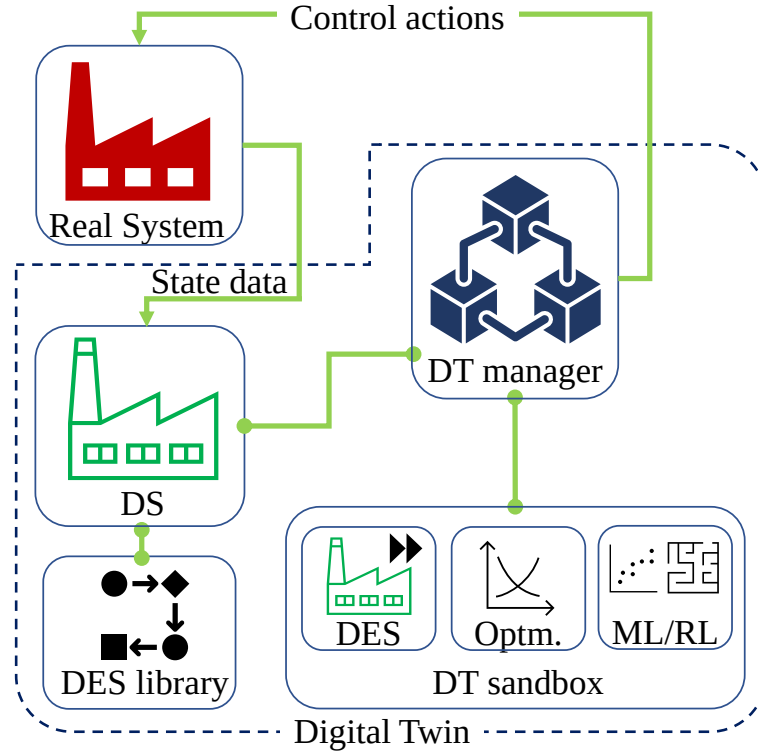


Figure 5.1 Digital Twin relationship with its corresponding real system

A DT needs to assess some configurational or operational modifications. Such modifications consider the current state of the real system. Real system's current state involves WIP information and equipment/operators state. Hence, simulations should not start by assuming an empty system or random initial state for equipment and operators.

5.5 Limitations of Traditional DES Integration

DES software packages allow model configuration and state initialization at beginning of simulation run. Arena simulation package can execute embedded Visual Basic for Applications (VBA) macros. This allows coded modules to automate model configuration which is normally executed manually through hard coding. Similarly, state initialization is achieved according to data acquired from the real system. Consequently, data-driven modeling is more suitable to facilitate the use of these two features.

In a data-driven model, the difficulty in achieving an automated model configuration and state initialization lies in properly structuring and translating the data according to model's requirements. As for a DS, such data are acquired from the real system and updated in real-time. Industrial data may have a format and a structure which are not compatible with

the operating DES model. Hence, the data in its raw form cannot be utilized for model configuration nor for state initialization.

After data structuring and translation, there is another obstacle to be addressed. The relevant information needs to be extracted from the translated data. For example, in semiconductor industry, a WIP cassette of wafers has to be identified whether it is a split of other cassette or not. These type of information are not necessarily available directly in the raw data. It is required for model's local and global variables. Hence, WIP initialization can not be completed without extracting such information from the translated data.

5.6 Proposed Approach

5.6.1 Semiconductor Manufacturing Systems

We applied our approach in a semiconductor manufacturing system. In such systems, semiconductor wafers are manufactured through a repeated cycle of several processes. The wafer design complexity determines the number of these cycles. The quality of wafers is measured in between processes in which wafers maybe reworked or scrapped.

Semiconductor wafers are produced in clean room environment. These manufacturing systems consists of several separated fabrication areas. A fabrication area includes different work areas in which same type equipment are setup. Same type equipment may differ only in qualification or mode of operation.

In this industry, wafers are transferred among different steps in lots or, cassettes, of a fixed number of wafers and certain sizes. The processes are called recipes. Each recipe requires certain equipment type and has specific settings. They may require certain pre/post-processing. This may include lot splits of rework and IF conditions. IF-conditions are executed through products' process flow according to measured parameters, in which lots are split into 'Then and Else' processing steps. Within the same equipment type, some equipment may have restriction against or dedication for specific recipes. This depends on production-related factors such as time since last maintenance.

Lots are processed according to equipment capacity. Some equipment allow processing of several lots having the same recipe. Other equipment allow only one lot or certain number of wafers. Operators are required to load/unload equipment at each processing step. According to their level of training, they qualify to operate certain equipment types.

Through a process flow, lots may be split at certain steps for a rework procedure or for an IF-condition procedure. Such splits may happen more than once for the same lot. In

these cases, splits of the same lot are rejoined at specific steps after finishing the required procedures.

Timers are special type of processes. A timer is a certain number of subsequent steps in which a lot must reach the last step within a certain time after finishing its first step. Otherwise, it may need to start over these steps or be scrapped if rework is not possible.

5.6.2 DES Modeling Scheme

A DT needs its operating DES model to be reconfigurable so that it follows its real system's possible modifications. This characteristic is essential for a DT to grant flexibility for its experimentation. Furthermore, a DT needs flexibility for decision making and execution within the model. These requirements are achievable through two modeling schemes which are data-driven and Agent-based modeling.

Given the mentioned requirements for a DES model within a DT, the model should follow system's behaviors only. The hardcoded part should solely contains logic modules that are as generic as possible. Such logic is mainly for agents' operation and products routing through model. The data-driven feature also extends to include agents operating logic as well. For example, heuristic dispatching rules are acquired from the data instead of being implemented within the model.

5.6.3 Data-Driven Modeling Implementation

In data-driven models, lots' flow is directed by the data through the model. By considering agent-based scheme, data also manage agents' working logic and partially control their behavior. In our use case, the input data are grouped into the following, as shown in Fig. 5.2:

1. Resources data
2. Processes data
3. Products data
4. Agents data

In DES, the resources provide processes or services to products. This includes equipment, operators, and transporters. In the example system, data for an equipment cover its equipment type, its capacity, as well as loading type, its capacity, restrictions or dedications for certain recipes or products, and uptime/downtime distributions or production counters. For

operators, data include their qualifications for different equipment, working shifts, and fabrication area. Similarly, transporters, which transfer lots among fabrication areas, are defined by their availability and capacity.

Processes are all procedures that lots should pass through to become finished products. In the example system, processes or recipes have related information including their equipment type by which they can be processed. Also, their processing times and average yield are among the recipes-related data. Other type of procedures involve processing a number of recipes within a certain time limit. Moreover, some recipes involve bonding wafers from different lots together. Hence, the lots should be ready together before starting such processes.

Each product has its own process flow that consists of its processing steps and specific pre/post processing procedures. The data include also products' orders information regarding quantities and due dates. These are used to calculate performance measures on which dispatching and order releasing depend.

The last group of data is model's agents data. In the example system, agents are responsible for order release, dispatching, and after processing procedures. The order release agents are responsible for releasing lots in the system at certain times according to the set strategy. This can include predetermined release times or probabilistic interarrival times for a push system. The dispatching agents dispatch the waiting lots at the queue of each equipment type according to the dispatching strategy. This can be a set of heuristic rules or an adaptive ML-based strategy. The after processing agents handle lots after being processed. They consider unloading lots from their equipment, exiting a cluster, rework or IF-conditions splitting/rejoining procedures, yield assignment.

In Fig. 5.2, the colored icons included in different data groups mark the link between these data groups within model's database. For example, recipes data are linked to equipment data as each recipe has its specified equipment type for its processing. This linkage is needed in case a certain indexing, different from system's original one, is used for processes, resources, and other system components. Hence, if a change is applied in any data group, all related data groups are automatically updated.

5.6.4 Agent-Based Modeling Implementation

Hybrid agent-based DES models employ agents to implement certain operational strategies within the model. These strategies cover product starting time or order release, allocation decisions, specific pre/postprocessing procedures, and any other decisions or procedures within the model. These procedures are normally implemented within models in process-driven

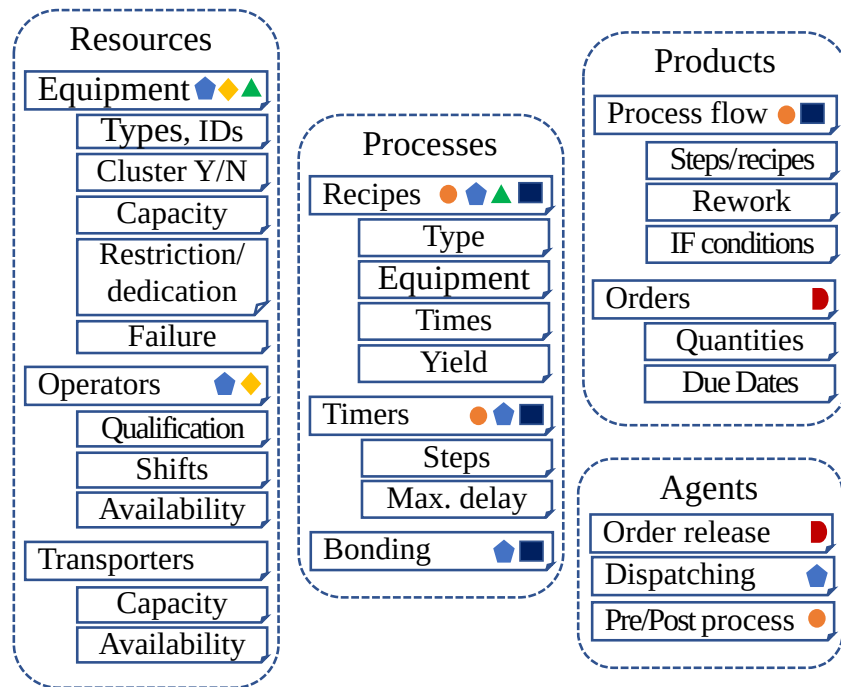


Figure 5.2 Data structure of a data-driven agent-based DES model for a semiconductor manufacturing system

schemes. This is where products are forced into a fixed predetermined flow.

The simulation model under consideration is based on the discrete-event simulation paradigm, which leverages discrete time steps to advance the model state as new events are generated in its event list. In addition to this, the model follows an agent-based modeling methodology, where logical entities, considered agents in the Arena's context, exist within the model and utilize specific logic to execute functions and operations.

The logic applied by agents can either be pre-programmed using heuristics, or trained using machine learning techniques to create a decision-making policy. The pre-programmed logic can be used to guide the dispatching of entities, while the machine learning-based approach can enable agents to learn and adapt to changing circumstances, resulting in more intelligent decision-making. This makes agent-based models more suitable to DT systems than non agent-based ones.

Fig 5.3 shows the proposed agent-based modeling scheme for the use case. The model consists of 4 main types of agents each has certain responsibilities. Agents behavior can be changed through the driving data in the database.

The first type of agents is for order release. Each product type has its own releasing agent.

This agent determines when to start a new lot from that product type. A simple strategy, for a DES model, is to use interarrival times or predetermined scheduled times. Another strategy is to employ a ML algorithm for making these decisions. This behavior can be changed easily by the DT through model's data. According to model's data, order release agents also assign or update model's local and global variables. Hence, the processing of the released lot can be started. In Arena, local variables are entities attributes, while global variables are accessible by all model's entities/modules.

The second type is preprocessing agents. These agents handle lots before starting their processing steps. They change model's local and global variables, gather lots that must be bonded, and do any required procedure before sending lots to wait for dispatching agents. In our case, only one agent is assigned for this task.

The third type is dispatching agents. Dispatchers apply the selected dispatching strategy and allocate equipment and operators for the waiting lots accordingly. They check some aspects of the current step of the selected lot before proceeding. For example, processing within a timer is only started if all equipment of the timer's steps are not failed.

The fourth type is postprocessing agents. These agents handle procedures that are done after the lot is processed. This starts by allocating operators for unloading equipment. Then, recipe yield is assigned for the processed lot. The need for lot splitting is checked also if a rework is required or an IF-condition is coming up within the process flow. Similarly, splits are held for rejoin if the current step is the rejoin step. Finally, lots are sent to preprocessing agents to process next steps or to send to exit the system if process flow is finished. In the example case, post processing agents are two agents. The first one is responsible for the mentioned tasks. The second one handles lots which have an exceeded timer limit.

5.6.5 Integration Scripts

As shown in Fig. 5.4, an integration library is introduced to the DT. This library contains scripts that convert real system's configuration and state-related data into model's format. Hence, the DT rebuilds its DES model and initializes its state to resume its main function. Similarly, when simulation is required by the DT, the model becomes ready with these features which serve the required experimentations.

We propose two main integration features:

1. Model reconfiguration
2. Model state initialization

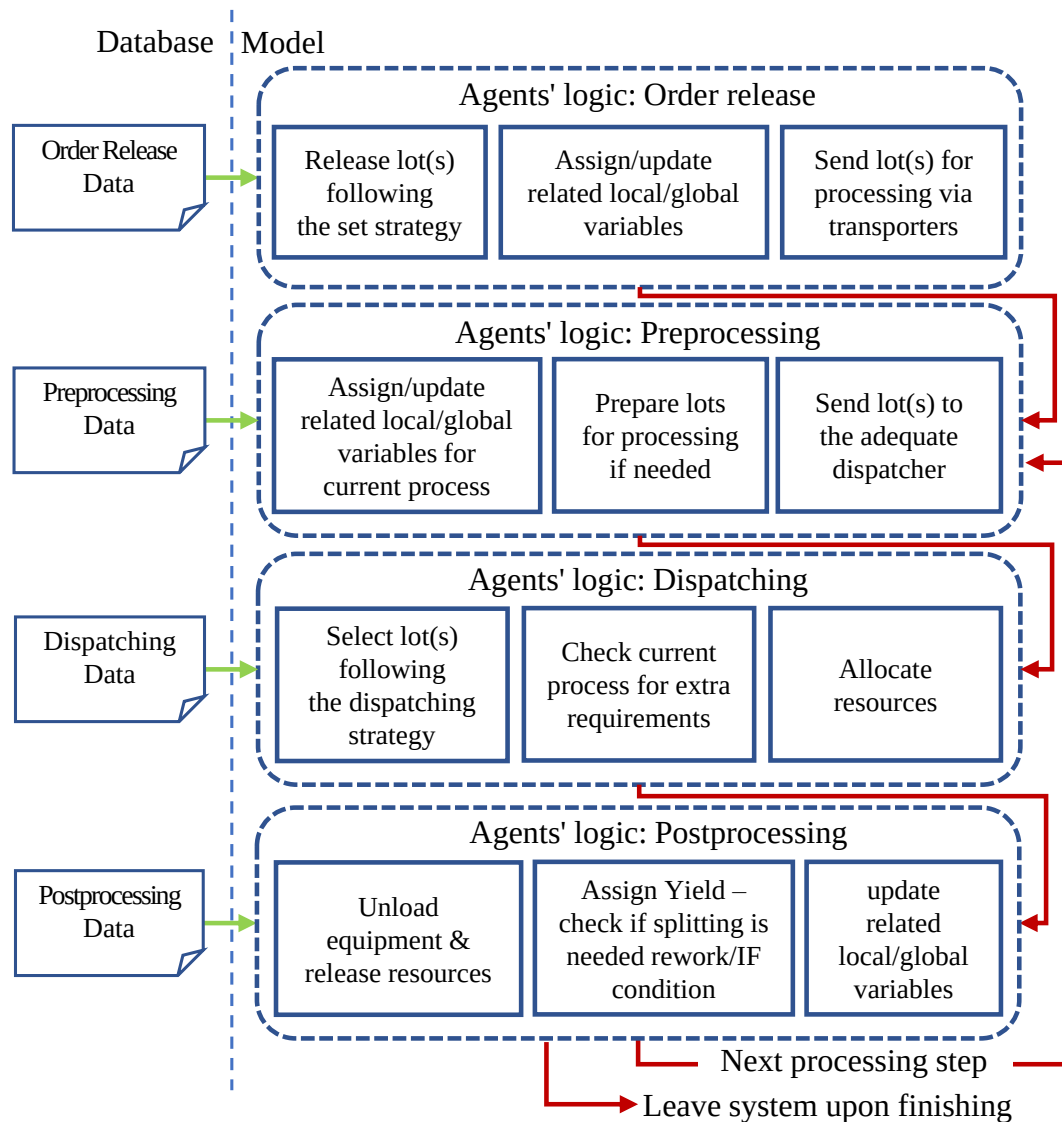


Figure 5.3 Agents operating logic through the DES model

Model Reconfiguration

The model reconfiguration feature allows the DT to reconfigure the DES model according to the database. The DES model's reconfiguration includes modules and configurations needed for a model to function.

As shown in Fig. 5.5, reconfiguration starts by a naming layer in which all modules are indexed and named according to the model database's naming convention. For example, equipment can be given names as: rEqm1, rEqm2,..etc. This step is essential as the reconfigured modules should follow their naming convention which is used through the database. In our case, this layer is included in the database file and is updated automatically according to the file's

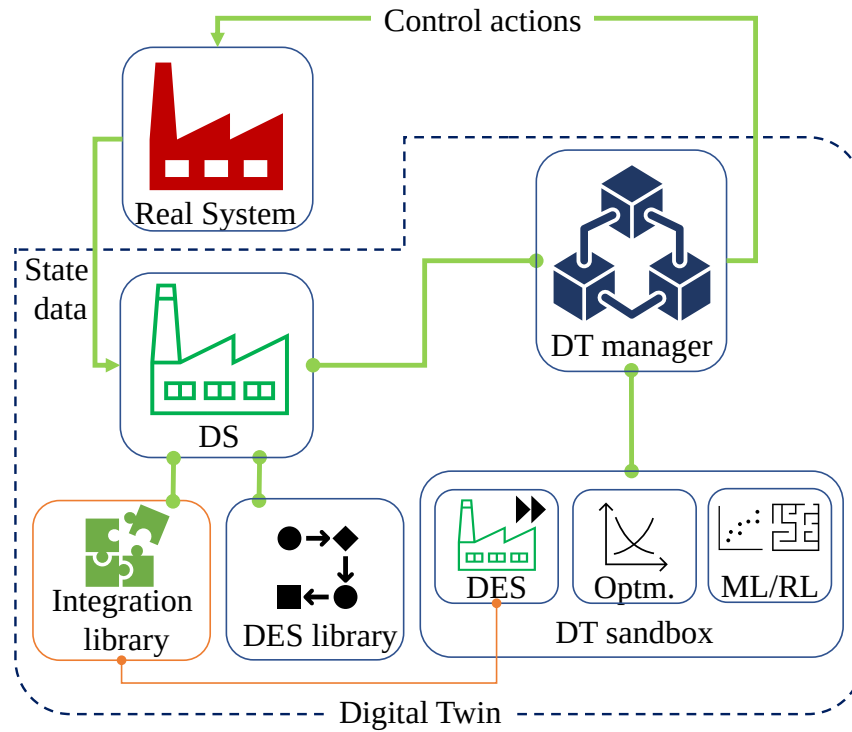


Figure 5.4 Introduction of the integration library within a Digital Twin

contents.

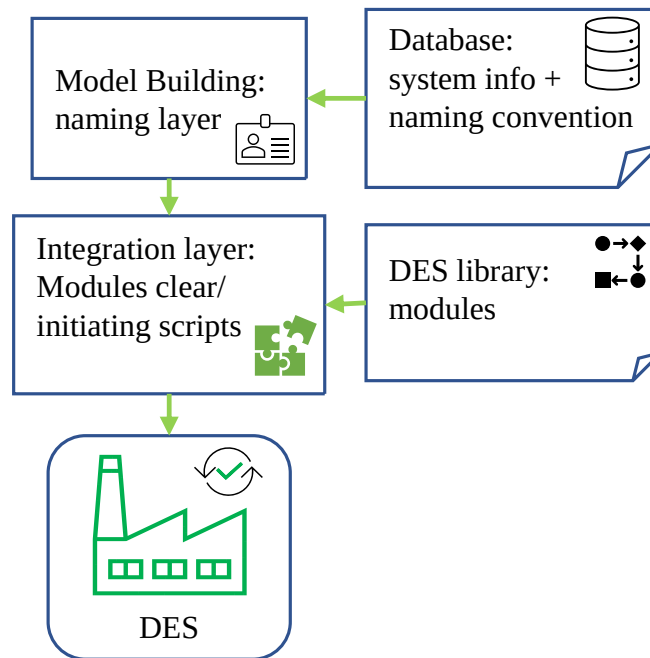


Figure 5.5 DES model reconfiguration steps

After the naming layer, the next step is done by the integration layer within the DES model itself. In our case, this layer consists of 2 main VBA scripts. The first one clears all modules that are going to be reconfigured from the model. The second one adds different modules according to their given name and indexes in the naming layer. These scripts are executed before starting the actual simulation, which is controlled through the data, so that the DT can let these scripts run when needed.

In Arena, VBA initializes its connection to the model and other external data files as shown in Algorithm 1. Hence, the rest of the scripts can access the data within the model and externally through an Excel file. The script that clears the modules passes systematically by all the modules of the same kind. For example, the VBA code, shown in Algorithm 2, is used to clear all resources modules for equipment within Arena. The "Sheet Name", "Named Range", "row", and "column" are following the structure of the data file which is used for storing the naming layer output. Similarly, all other modules, that needs to be reconfigured, are cleared by giving their module names' prefixes that are acquired from the naming layer.

Algorithm 1: Arena model and Excel file access initialization via VBA

```

1 'Part1:
2 Dim our_model As Model
3 Set our_model = ThisDocument.Model
4 Dim FileToOpen As String
5 Dim ArenaDir As String
6 Set XL = GetObject("", "Excel.Application")
7 ArenaDir = Mid(our_model.FullName, 1, Len(our_model.FullName) -
   Len(our_model.Name))
8 FileToOpen = ArenaDir & "input_data.xlsx"
9 XL.Workbooks.Open FileToOpen

```

After clearing the modules to be reconfigured, these modules have to be rebuilt according to the new configurations following the database. The VBA code, shown in Algorithm 3, is used to create and reconfigure all equipment resources. The counter i is incremented up to the number of the equipment that is acquired from the database. The other equipment's information are assigned following the same manner. Similarly, configuration is done for all other modules.

Model State Initialization

For experimentation, the DT needs to start a simulation from the real system's current state or a specific state in general. In this case, the model state initialization becomes an essential

Algorithm 2: Clear resources modules for equipment

```

1 Dim eqm_i As Variant
2 eqm_i = 1
3 Dim eqm_prefix As String
4 eqm_prefix = XL.Workbooks(1).Worksheets("Sheet Name").range("Named
   Range").Item(row num, column num)
5 while our_model.Modules.FindByData("Name", eqm_prefix & eqm_i) > 0 do
6   ModuleIndex = our_model.Modules.FindByData("Name", eqm_prefix & eqm_i)
7   Set MyModule = our_model.Modules.Item(ModuleIndex)
8   MyModule.shape.Selected = True
9   our_model.ActiveView.Selection.Delete
10  eqm_i = eqm_i + 1

```

Algorithm 3: Configure resources modules for equipment

```

1 for i = 1 To XL.Workbooks(1).Worksheets("Sheet Name").range("Named
   Range").Item(row, column) do
2   Set MyModule = our_model.Modules.Create("BasicProcess", "Resource", i, 1)
3   MyModule.Data("Name") = XL.Workbooks(1).Worksheets("Sheet Name")
   .range("Named Range").Item(row, column) & i
4   MyModule.Data("Type") = "Fixed Capacity"
5   MyModule.Data("Capacity") = XL.Workbooks(1).Worksheets("Sheet Name")
   .range("Named Range").Item(i, column)
6   MyModule.Data("StateSetN") = XL.Workbooks(1).Worksheets("Sheet Name")
   .range("Named Range").Item(row, column)
7   MyModule.Data("InitState") = XL.Workbooks(1).Worksheets("Sheet Name")
   .range("Named Range").Item(row, column)
8   Next i

```

feature for a DES model. Having this feature is challenging as the real system's state can not be directly used to generate a corresponding state in the model. In DES, there are several local and global variables that depend on the model's state. For the simulations where systems start empty, these variables are developed automatically along with the model's state's update. To start a simulation from a certain state, these dependent variables must be evaluated and assigned. Hence, that state becomes fully defined for the model to start with.

In our case, local variables are identification numbers, product ID, step ID, and all other identification information that are related to the lot at its specific processing step. Global variables include information that involve all processing lots and resources, for example, equipment state, WIP information, and all information that are shared among model's agents and entities. The state initialization procedure is depicted in Fig. 5.6. The translation layer extracts representative information from real system's state raw data. In our case,

the translation layer is a Python script that interacts with the database to generate the local and global variables corresponding to the naming convention. According to the step, identification numbers, and time stamps, a lot step is fully identified. This translates into the following:

- Step ID, recipe ID following product's process flow
- Equipment type ID following model's database indexing
- Type of step: normal process, within a rework,...etc

The translation layer's output, which is state initiation data, is a full definition of the real system's state. This definition corresponds to the one within the model including local and global variables values. Consequently, the integration layer utilizes these data, creates lots, and assigns variables values. Finally, the integration layer sends the created lots to their location within the model corresponding to their step type.

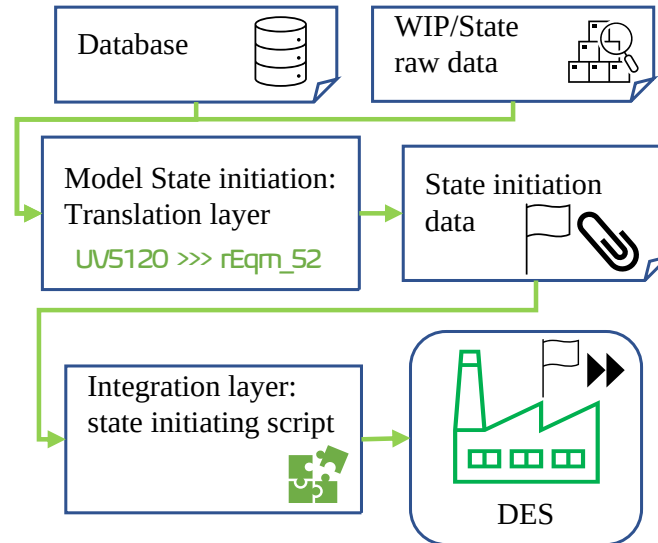


Figure 5.6 DES model's state initialization steps

In our case, the integration layer is a VBA script that acquires the state initiation data. Then, it assigns the variables' values accordingly within the model at the beginning of the simulation run. Algorithm 4 shows a part of the integration script. Attributes values are acquired from the translation layer and assigned to the created entities. Hence, they represent their real correspondence in the real system. These created entities are then sent through the model to locations according to their state, which is referred to as the initialization type. This includes waiting in queue, processing, and all other possible locations for an entity.

Algorithm 4: Assign attribute values for a created entity and sent it to the location within the model corresponding to its current state

```

1 'Assign user-defined attributes
2 for attr_i = 1 To num_of_attributes do
3   our_siman.EntityAttribute(our_siman.ActiveEntity,
   our_siman.SymbolNumber(attributes_names(attr_i))) =
   attributes_values(attr_i)
4   Next attr_i

5 'Insert in the entity according to its initialization type
6 our_siman.EntitySendToStation our_siman.ActiveEntity,
   our_siman.SymbolNumber("station_WIP_" & Split(initialization_type, ".")(0))
7 Next attr_i

```

5.7 Use Case Experiment

We applied our proposed approach in semiconductor industry use case. In this scenario, a DT is to evaluate system's performance when commissioning a new product within a system that is already producing a certain product. We name the new product as Product 2, while the other product is named Product 1. This experiment is done at two different settings. The first setting is starting an empty simulation. The second setting considers some WIP and equipment state information and initializes the simulation accordingly. Both products are processed through 23 different equipment stations that have 23 queues for their waiting lots. More details for the simulation model are found in Appenix.A.

Before the beginning of the simulation, the integration layer updates the configuration of the model by its corresponding of the real system if different. If a model initialization is required, the integration layer acquires the actual WIP data and translates it according to model's variables. In our experiment, the WIP data are provided in the second setting.

Fig. 5.7 shows the average queue waiting time at each equipment for the two settings. Fig. 5.8 shows the average total time in system and average WIP for products. When starting empty, the waiting time for queue 6 is almost 2.5 times its duration when the system is initialized. The opposite happens for queue 8. The results indicates the clear deviation of these performance measures for the two settings. Hence, DES's state initialization must be considered for experimentations within DTs.

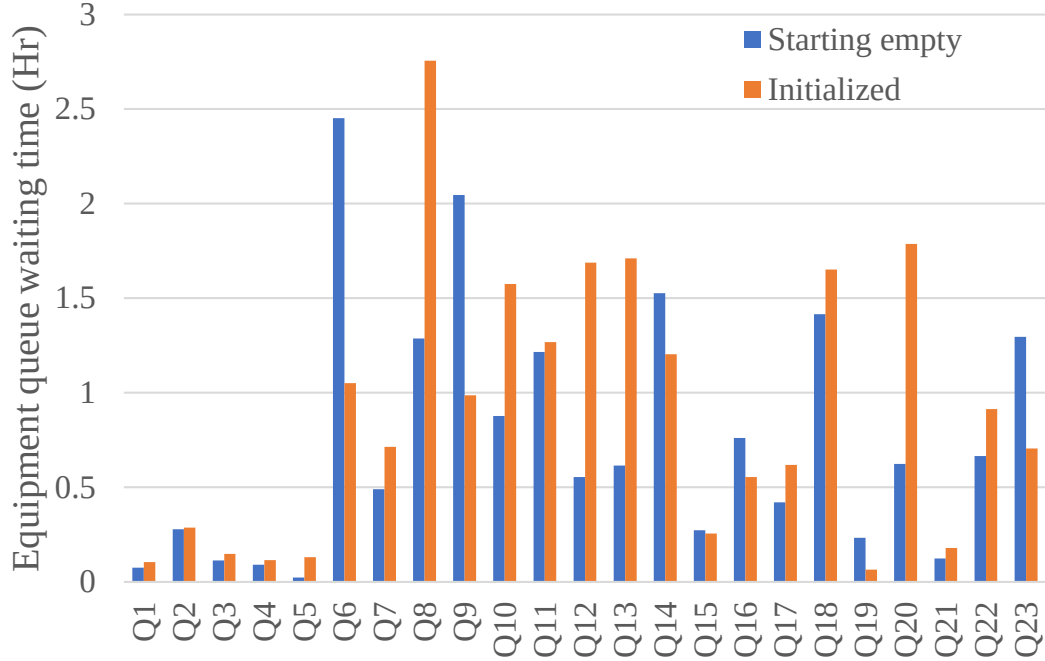


Figure 5.7 Average queue waiting time at each equipment with and without state initialization

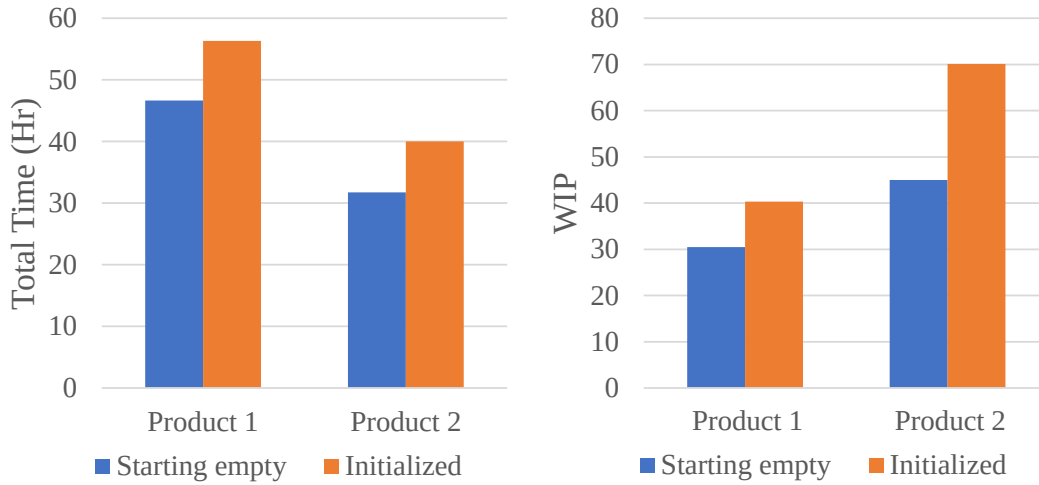


Figure 5.8 Average total time in system and average WIP for products with and without state initialization

5.8 Conclusion

In this paper, we proposed an integration approach and building aspects for a Digital Twin based on Discrete-Event Simulation model. We presented our approach with the aid of a semiconductor manufacturing system as a use case. We presented the limitations of the

traditional DES modeling to integrate within a DT. We built a data-driven and agent-based simulation model within a DT framework. The proposed approach allowed us to build main procedures, to structure the data, and to establish a model that is governed by agents and data-driven scheme. We introduced an integration layer that provides two essential features: reconfiguration and state's initialization. These features provide the simulation models the configurability and integrity required by DT. Hence, it extends the usability of current DES for DTs within Cyber-Physical Systems.

Acknowledgement

This work is funded through the Mitacs Accelerate program, Grant No. IT01632, in partnership with Teledyne DALSA Semiconductor, a business unit of Teledyne Digital Imaging, Inc. The authors would like to thank the industrial and information technology teams at Teledyne DALSA semiconductor for their great support and collaboration.

**CHAPTER 6 ARTICLE 3: SIMULATION AND DEEP REINFORCEMENT
LEARNING FOR ADAPTIVE DISPATCHING IN SEMICONDUCTOR
MANUFACTURING SYSTEMS**

Ahmed H. Sakr, Ayman Aboelhassan, Soumaya Yacout, Samuel Bassetto

Published in:

Journal of Intelligent manufacturing

October 8, 2021

6.1 Abstract

Fabrication areas in semiconductor industry are considered one of the most complex production systems. This complexity is caused by the high-mix of products and end-user market-based demands in that industry. Its dynamic and challenging processing requirements affect the handling capabilities of traditional production management paradigms. In this paper, we propose an application for dispatching and resources allocation through reinforcement learning. The application is based on a discrete-event simulation model for a case study of a real semiconductor manufacturing system. The model is built using both data-driven and agent-based approaches. The model simulates the various processing aspects that are present normally in these complex systems. The model's agents are responsible for dispatching tasks and allocation of the different system's resources. They employ Deep-Q-Network reinforcement learning. They learn simultaneously through the model execution. An independent Deep-Q-Network is trained for each agent. The model provides the training environment for the agents in which their decisions are applied and assessed for their adequacy. Our formulation of the environment's state and the reward function for the learning algorithms creates cooperative decision-making policies for the agents. This results in improving the global performance of the whole system, and the performance of each agent's resources. Our approach is compared to heuristics-based strategies that are applied in our case study. It achieved better production performance than the currently applied strategy.

Keywords

Dispatching, Complex production, Deep reinforcement learning, Discrete-event simulation, Agent-based

6.2 Introduction

Recent growth of digital connectivity technologies has increased the competitiveness among production systems [6]. Industry 4.0 introduced these communication and information technologies interconnecting the production resources. Such resources collect and transfer production and performance data continuously. This facilitates the decision-making decentralization that has become one of the strategic responses of the competitive manufacturing firms [2, 3]. These technologies provide opportunities for identifying intercorrelations and insight through analyzing data, hence supporting decision-making process.

Data insight tasks are difficult for analysts [24]. This problem is apparent in case of complex

production systems such as pharmaceuticals, mining, and semiconductors. A product or production is considered to be complex if it is subjected to variable customers' demands with recurrent changes in product's specifications and/or production flow. This scheme of production confronts many manufacturing enterprises and requires internal logistic flexibility to satisfy their customers' needs [9].

In semiconductor industry, there are a high-mix of products and a market-based demand [69]. Different products share the same production resources and they have intersecting flow through fabrication areas, various split/rejoin routes, and waiting-related quality issues. This creates different sources of complications for the management tasks, and hinders proper decision-making for capacity planning and delivery times estimation. Failure to properly manage such systems results in excessive amounts of waste, non-value added times and scrap, and hence costs. The waste in such systems is normally found due to improper dispatching strategies and inadequate resources allocation. The effect of having high waste is amplified due to the repetitive processing steps that are required [84].

For complex productions systems, Discrete-Event Simulation (DES) is considered a decision support tool for performance assessment and experimentation [18]. DES captures the dynamics of the system along with its stochastic nature. This is the foundation for model-based decision-making support within the Industry 4.0 concepts that enables such models applicability in real-time [15]. This extends from normally applied heuristics solutions to imbedded artificial intelligence methodologies such as Reinforcement Learning (RL). In this context, DES provides a virtual representation of the system as the required learning environment for RL algorithms. Hence, these algorithms interact with that virtual environment and learn to make practical decisions in different simulated cases.

In this paper, we develop an application for model-based decision-making for a case study of a complex semiconductor manufacturing system. Our application involves establishing a DES model for a real system. The model is designed and built to adopt both the dynamicity of system's parameters, and the applicability of learning methodologies for decision-making. These functionalities are realized through the data-driven and agent-based modeling. The data-driven modeling provides the flexibility that is essential upon undergoing configurational or productional modifications. The agent-based modeling facilitates the creation of artificial agents and the implementation of their decisions. These agents are required for decisions-making through the simulation. They are trained to reach their operational goals through making decisions for both dispatching and resources allocation. They consider local objectives, resources' level, and global objectives on system's level. The agents are trained within the system's virtual environment that is realized by the DES model. The environment's state

definition is formulated so that it minimizes the effect of the continuously changing system's parameter on the adequacy of the learnt agents' performance. This paper contributes to providing a comprehensive approach to apply RL-based dispatching and resources allocation for semiconductor production systems in a cooperative agents environment. The state information and the reward formulation are done considering challenging decisions that are frequently required in this industry. We present a systematic DES model building approach for semiconductor production systems to create the required training environment.

The rest of the paper is organized as follows: Section 6.3 presents a literature review for production management-related problems in complex production systems and the available solutions through simulation, optimization, and reinforcement learning. Section 6.4 states the problem in the semiconductor industry as an example for a complex production system. Section 6.5 presents our methodology for creating a simulation environment, and employing the RL for solving the dispatching and resources allocation problems. Section 6.6 presents a semiconductor manufacturing system case study, and our RL results compared to the currently applied heuristics-based strategy. Finally, Section 6.7 provides a conclusion for our works and results.

6.3 Literature review

6.3.1 Optimization with simulation models

DES has been recognized as a stochastic and dynamic tool for analysis and performance improvement in complex production systems [18]. DES answers key operational questions such as throughput assessment, debottlenecking, and resources utilization. Answering these questions enables the decision-makers to apply effective operational modifications [28]. Reményi and Staudacher [29] presented a systematic simulation-based approach for the identification of suitable scheduling rules in aircraft maintenance sector. They addressed cost prediction and turnaround time reduction. Their approach showed the significance of decentralized scheduling rules on logistical targets, such as on-time delivery and orders' creation time reduction. Ghani et al. [30] used DES for modeling a production assembly line in the automotive industry. They developed an algorithm for energy usage optimization based on dynamic property adjustments such as accelerations and motion types of machines' components. In their research, the DES is used to compare different adjustments, and assess the production performance. In their study, performance indicators were production rate, cycle time, and shift patterns. They were optimized along with the minimization of energy consumption per product. McDonald et al. [31] addressed the effect of varying demand levels

on a motor manufacturing cell. They developed a DES using the commercial package Arena simulation to determine production capabilities and potential problems for various demand levels. Their results indicated potential bottlenecks for each level of demand. Korytkowski et al. [32] optimized the dispatching rules for a commercial printing production system. They applied simulation-based heuristics for optimal dispatching rule allocation. Their model is implemented using DES and optimized using a genetic algorithm. Their main outcome is system efficiency improvement through properly managing the sequence of orders. They improved the cycle time, and reduced orders' tardiness and queues size. Edis et al. [85] developed a DES model for addressing lot streaming problem. This is for a stochastic job shop environment with multiple products. They analyzed different performance measures for different transportation queue discipline and number of subplots for each product. Their proposed heuristics slightly improved some performance measures.

Traditional optimization algorithms may find the optimum strategy for management tasks such as dynamic scheduling or dispatching. However, such solutions may not stay optimum for long periods due to being based on average values of simulated performance measures. In addition, they may be optimal only for certain system's configuration and less optimal for another ones. In this case, new simulation runs are required to provide new optimum solutions. Typically, such runs take a long time, especially for complex production environments [86]. Also, it is impractical to rely on optimization-based decision-making for real-time implementation. This is due to the long computation times, and the fact that such algorithms do not learn the environment.

6.3.2 Reinforcement learning with simulation models

Adaptive dispatching strategies are used to overcome the challenges in complex production systems [59]. Meta-heuristics optimization approaches are traditionally applied to solve problems with a limited computation time and incompatible complexity. Such problems belong to the NP-complete problem domain that are computationally expensive to solve. For dispatching, rule-based heuristics are widely applied in such manufacturing systems. It is known that no rule outperforms all others permanently under any objective. Also, multiple rules may contradict each other [60]. The optimal dispatching is system state-dependant. In this case, learning a strategy for solving the dispatching problem is a promising approach [61].

In RL, an agent is trained within an environment. It learns to take practical actions that provide good results. This is done by exploring and learning a policy for action selection instead of being supervised or corrected. The RL is most suitable when there is no knowledgeable supervisor. Thus, the agent needs to learn through interaction with the environment. This

is achieved by exploiting its learnt experience and exploring new strategies [21].

Kuhnle and Lanza [65] applied RL in production planning and control of a Cyber-Physical System (CPS) in which physical resources are monitored and controlled through computer-based algorithms. They addressed the decisions that are related to both order dispatching and maintenance management. Their agent yielded better performance for utilization and throughput time as compared to reactive and time-based maintenance strategies. It also learnt how to perform maintenance closer to failure and how to predict failure time. Xanthopoulos et al. [87] proposed a RL-based maintenance control policy for a manufacturing system. They defined the reward function as a sum of two conflicting objectives. These objectives are the inventory level and the number of backorders. Their solution consisted of achieving an optimal trade-off between maintaining high service levels and keeping the lowest possible inventory. The proposed policy outperforms the examined production and maintenance policies including Kanban, condition-based maintenance, and periodic maintenance. Rocchetta et al. [88] developed a RL framework for managing the operation and maintenance of power grids. They identified the optimal actions for maximizing the expected profit. Their framework outperforms expert-based solutions for grid management.

In this research, we extend the previously mentioned applications to involve multi-agent environment in a complex production system. We address real challenging decisions that affect the performance of these systems. In our environment, agents learn together to cooperate and to improve their local performance and the global performance of their system. Each agent is responsible for decisions of its assigned equipment. This decentralized system provides agility and reconfigurability that are hard to achieve by a centralized strategy [46]. This enables each agent to learn an adequate dispatching policy according to its equipment aspects including loading/unloading operation, capacity, and processing time. The training environment for our agents is realized through a DES of a real semiconductor manufacturing system. We enrich the agents policies adequacy through a generalized formulation of the environment’s state information, and a representative reward function. The full details of our approach are presented in Section 6.5.

6.4 Problem statement

6.4.1 Semiconductor industry overview

In semiconductor industry, the wafers are processed through repeated cycles of several processing steps. Through these steps, thin layers of silicon are built on the wafers to form the required integrated circuits [18, 89]. Semiconductor factories consist of several clean-room

environment fabrication areas. Each fabrication area consists of several work stations. These stations gather several equipment of the same type and capabilities.

A lot consists of a fixed number of wafers, 25 wafers in the use case, of a certain wafer size which are transferred in cassettes among processing steps. Lots are processed according to the equipment's processing scheme. These schemes include: (1) processing one wafer at a time, (2) processing the whole lot or more than one lot at once, and (3) processing part of the lot, certain number of wafers, at once according to the equipment's capacity. In the later, the lot is split and rejoined after processing. Some equipment are clustered in one single frame forming a cluster. Such clusters have an inside robot manipulator that load/unload the equipment. Accordingly, it processes several steps with less need for an operator to load/unload the cassettes. Normally, the operators are qualified to operate some equipment types according to their level of training and experience.

This industry has both characteristics of discrete manufacturing and process manufacturing. It relies on recipes as in process manufacturing, while having the complexity of discrete manufacturing, which are characterized by the following [69,90]:

- Long cycle times
- Make-to-order or Engineer-to-order
- Complex process sequences
- Equipment reconfiguration for different recipes
- Operators various qualifications
- Equipment recalibration

6.4.2 Semiconductor processing aspects

Each product's process flow consists of a set of recipes that are assigned according to its design. Overtime, recipes are subject to change, addition or removal. This is due to the nature of engineer-to-order in this industry. Hence, manufacturing strategies are continuously improved for better yield and product's quality. These changes may include some restrictions for specific equipment to process certain recipes or products until maintained or re-qualified.

Some products include timers through their process flow. For example, if there is a timer through steps A-B-C-D, upon finishing step A, the lot should reach step D within the timer's duration. If not, the lot will have to be scrapped or at least reworked through the same steps.

Through timers' rework, the lot has the same timer's duration limit. The rework within timers can not be done more than once for the same lot. Furthermore, lots are subject to split in several situations. Some steps normally have to be reworked for part of the lot. The part that needs a rework is sent for reworking and the other part continues normally until a certain rejoin step. IF-conditions are other type of splits which are decided through certain parameters' measurement. In IF-conditions, the lot is also split into 2 parts and processed according to *Then* and *Else* processing steps. A rejoin is done after the IF-condition's splits processes are finished.

6.4.3 Dispatching tasks in semiconductor industry

There are different heuristics-based dispatching rules that are normally applied in semiconductor industry. These rules include First-In-First-Out (FIFO), Earliest Due-date (EDD), and earliest-timer limit. The EDD is often replaced by the Slack Ratio (SR) in this industry [91, 92]. The SR relates the remaining time till each order's due-date ratio to the remaining quantities to be produced. Normally, the lot with the nearest limit is dispatched first according to the earliest-timer limit rule.

There is no rule that outperforms all others. Practically, various rules are applied simultaneously with certain hierarchy. For example, timer limits are usually of a higher importance than slack ratio or arrival time of waiting lots. In case of equal or no timer limits, then the next rule SR is applied. Due to high-management decisions, relative priorities may be assigned to different products and used as an extra rule within the hierarchy.

Given these strategies, the products flow becomes subject to many stochastic and dynamic alternatives. These alternatives depend on the Work-In-Process (WIP) products, managerial decisions, and continuously changing customers' needs. Best design and selection of dispatching-rules become hard due to the altering productional configurations.

6.5 Methodology

6.5.1 System modeling overview

In RL application, the real system's model plays an important role for providing the training environment. The environment acts as an experimentation platform that triggers the agents when actions are needed. It passes the system's state to the agents, so that they select the proper action accordingly. Upon action selection, the environment performs the action, assesses its adequacy, and sends its feedback to the agents which is used for learning.

The selection of modeling technique should correspond to the system's operational behaviors. Due to the system's dynamics, DES is chosen since it handles both stochastic and dynamic aspects.

We combine the approaches of data-driven and agent-based modeling for our DES model. The data-driven modeling allows system's configuration and parameters to be defined by an input data file instead of being hardcoded into the model. For instance, the input file includes the number of different products, products flow, and the other processing-related details. Data-driven models are more flexible to accept configurational changes while keeping the model compact and modular. A compact model permits adopting complex operational logics and, more importantly, controlling logics such as dispatching in these systems. The data-driven modelling scheme allows advanced functionalities for DES, for example, automation of model's parameters updating and synchronizing with its corresponding real system.

Agent-based modeling employs control agents for taking actions. The actions are taken based on a pre-defined set of rules, optimization algorithms, or machine learning algorithms. Generally, these actions are used within the model throughout its execution for tasks such as seizing/releasing a resource, moving a product from one step to the next, and other similar tasks.

In our model, agents act as dispatchers and resources allocators. They dispatch the waiting lots and allocate the equipment. They have local and global objectives regarding the system's Key Performance Indicator (KPI)s. The local objectives are related to the equipment station that each agent is responsible for. They include resources' utilization and queue's waiting time and number in queue. The global objectives are related to the system as a whole. They include other system's KPIs such as products lead-time and timers-related junk/rework quantities.

6.5.2 Simulation model in depth

Input Data

As mentioned, the model is data-driven in which its behavior depends primarily on the input data. We structured the input data file while bearing in mind the different operational aspects that the model should simulate. These operational aspects include processing steps, pre/postprocessing tasks, and all the various manufacturing aspects that normally exist in a semiconductor manufacturing firm. The main groups of our input data structure are:

- Products basic data including orders, quantities, and deadlines

- Process flow for each simulated product
- Recipes information and specific details for some recipe types such as bonding recipes
- Special processes information such as IF-conditions and rework
- Timers and all their related information
- Equipment information and their restrictions/dedications for specific recipes or products
- Operators information and their qualification for different Equipment

The products' basic data identifies each product and assigns a unique ID for each product. This product ID is used within the model to access other information that are related to the product. Other data include order release strategy, orders due-dates and quantities, basic priorities assigned for each product by the factory's management.

The process flow includes the processing steps for each product and all its details such as: recipe name, fabrication area, split/rejoin for possible rework procedures or IF-condition, and alternative recipes if any. Recipe data defines the equipment type, recipe average yield, processing times, and minimum delay required after each recipe if any. IF-conditions and rework data include the historical splitting probability for each wafer within the processing lot. Special recipes or procedures such as timers and bonding recipes are defined among the recipe information group.

Equipment are defined as groups of same equipment type. The equipment type defines the group which may contain one or more equipment, each defined with a unique ID named Eqm ID. The equipment can be clustered with other equipment in a single cluster. In this case, another ID is assigned for the whole cluster. For all equipment, the capacity, type of loading batch, and speed factor of processing are defined for each equipment. Uptime/downtime distributions are derived from equipment failure history. Equipment restriction/dedication are defined for some recipes or products. Among a group of equipment, a recipe or product may be restricted from processing by a specific equipment.

Human operators are also modeled in the system. The operators are responsible for loading/unloading lots into the corresponding equipment. Operators information define their working shifts, fabrication area, and qualification for operating different equipment types.

The data file groups are dynamically linked to propagate the effect of any modification among groups. This linking is important to avoid the possible inconsistencies or logical mismatching among the data. Such discrepancies affect the accuracy of the simulation results.

Figure 6.1 shows the data flow among the simulation model, the input data structure, and the real system's database. As shown, the main advantage of data-driven models is that it is possible to link the model's input data to the system's database. This insures that the model represents the most recent configuration of the real system.

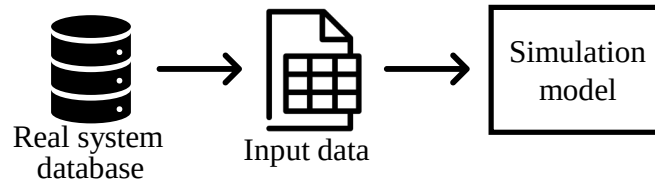


Figure 6.1 Data flow in data-driven DES

Model components

The simulation model consists of a number of sub-models which perform certain functionalities. These sub-models handle routing and logical processing in the system. The sub-models are grouped into products-related and resources-related. The products-related sub-models are responsible for products routing and processing through the model. The resources-related sub-models handles system's resources availability. They simulates equipment up-time/downtime and operators working shift and possible random interruptions that take place normally for operators.

Products-related sub-models simulates the following functionalities:

- Releasing orders and assigning lots' attributes
- Routing lots to the equipment's queues
- Handling timers initiation
- Agent-based dispatching and resources allocation
- Processing the lot according to the equipment batch type
- Unloading equipment and batching split lot's wafers if any
- Assigning yield and scrapping unqualified wafers
- Routing lots' wafers for rework if needed, and rejoin after rework
- Routing lots' wafers for IF-condition splits if any

- Sending lots for their next processing step according to the current and next fabrication area

Figure 6.2 shows an overview for the products-related sub-models as implemented in our model. For timers starting, the equipment availability is checked before the start of processing. Timers are normally handled according to their tightness with respect to the processing time within their steps. A timer is considered tight if it has a relatively long processing time among its steps with respect to its maximum timer limit. If the timer limit is tight, equipment are held in advance to ensure quick processing for the lot. Holding equipment in advance is not permitted in our RL approach even if the timer is considered tight. Additionally, dispatching agents, at the timer starting steps, decide when to start a lot within a timer.

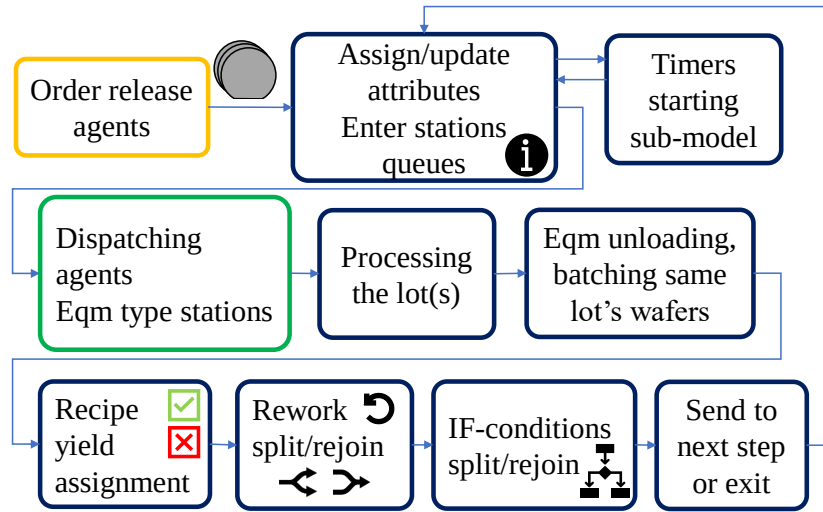


Figure 6.2 Products-related sub-models overview

Agents

There are two kinds of agents in our model. The first are dispatching and resources allocation agents. They learn, through RL, to select practical decision for lot dispatching and allocating equipment. The second are pre-programmed agents. They execute set of logical rules to simulate some functionalities such as handling operators working shifts and equipment uptime/downtime.

The main difference between dispatching agents and pre-programmed agents is the applied decision-making policy. The policy for dispatching agents is learnt through the simulation

execution, while policies of pre-programmed agents are pre-defined. The model rewards the dispatching agents according to the adequacy of their previous decision. The reward is calculated through different systems KPIs. Figure 6.3 shows how dispatching agents interact with their environment. According to the current state s_t of the environment, each agent select actions a_t . The environment then applies the action and feedback the agent with its reward r_t . The state representation and the reward function are discussed in details in Section 6.6.

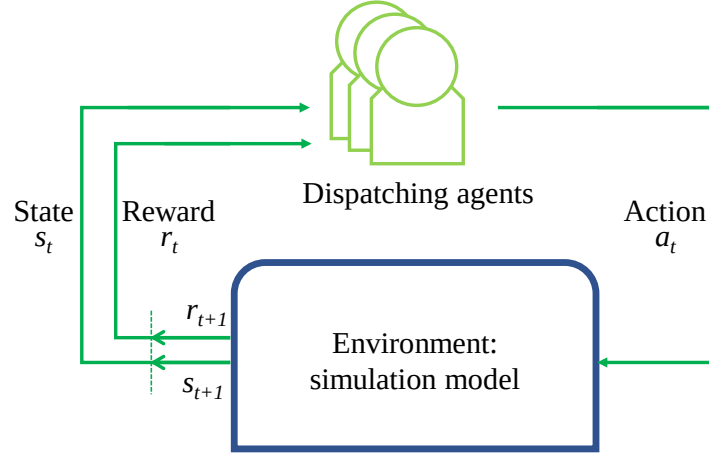


Figure 6.3 Dispatching agents learning process

6.5.3 Deep-Q-Network

Deep-Q-Network (DQN) is a RL algorithm that uses artificial neural networks, as Q-function approximator, to predict the expected reward, Q-value. This Q-value is calculated for each action for a given state according to the learnt policy π , $Q^\pi(s|a)$. We can define the optimal Q-function $Q^*(s|a)$ as the maximum reward that can be obtained following the optimal policy. The optimal Q-function follows the Bellman optimality equation as follows [93]:

$$Q^*(s|a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s'|a')] \quad (6.1)$$

where γ is the discount factor with a value in $[0,1]$. This value discounts the future rewards with respect to the immediate reward. A value of $\gamma = 0$ means that the agent focuses only on the immediate reward rather than the delayed future ones. From Eqn. 6.1, the optimal policy Q^* provides the action a which has the maximum Q-value given a state s . The Q-value represents the sum of the immediate reward r and the discounted future rewards obtained by following the optimal policy Q^* given the expected next state s' . Through the Bellman

optimality equation and iteratively updating the policy, the Q-function converges to the optimal one [93]. In DQN, the neural network with parameters θ , weights, is trained to approximate the Q-function, i.e. $Q^*(s|a) \approx Q(s, a; \theta)$. The loss function, in neural network training at step i , is minimized. The loss function L_i is:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \rho} [(y_i - Q(s, a; \theta_i))^2], \quad (6.2)$$

where y_i is defined as: $y_i = r + \gamma \max_{a'} Q(s', a'; \theta_{target})$. ρ represents the probability of transition from state s to s' as collected from the environment, the simulation model in our case. To stabilize the behavior of DQN, 2 networks are used for calculation of the loss, evaluation network and target network. The target network's parameters θ_i are not updated every learning iteration unlike the evaluation network. The target network's parameters are replaced by the evaluation network's parameters every replacement iteration.

In DQN, agents apply a technique which is called experience replay. In experience replay, the agent's experience, at each time of taking action, e_t , is stored in the replay memory $\mathcal{D} = e_1, e_2, \dots, e_N$. The agent's experience consists of the state-action-reward and the subsequent state as: $e_t = (s_t, a_t, r_t, s_{t+1})$. Through the Q-learning updates, a batch is drawn at random from the replay memory $\mathcal{D}$. This avoids having a highly correlated sequence of states, as training samples should ideally be independent.

Figure 6.4 shows how agents learn their policies through their 2 networks and their environment. The environment is provided by a DES model where agents' actions are applied and rewards are assigned accordingly.

Upon requiring an action, the agent may exploit its policy or explore a random action according to a value ϵ which varies in the range $[0, 1]$ representing the probability of taking a random action. This enhances the agent's learning performance through exploring different actions.

6.5.4 DQN for dispatching and resource allocation

Figure 6.5 shows how the dispatching agent handles the waiting lots through the simulation model. The agent acts when there are waiting lots in its station's queue. The agent first finds an idle qualified operator, in the same fabrication area, and an idle equipment among its station's equipment. Upon having an idle qualified operator and an idle equipment, the agent checks the waiting lots that are processable given the available equipment. A lot may not be processable due to an equipment restriction and/or dedication to another equipment, and if it is at a timer starting step and at least one of its next steps' equipment is down.

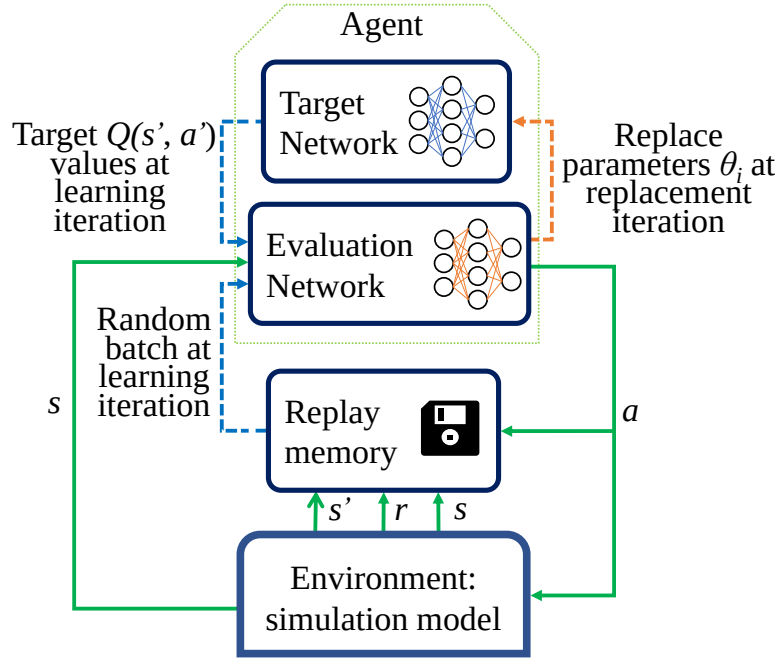


Figure 6.4 Agents learning process through DQN algorithm and its virtual environment

If non of the waiting lots is processable, the agent checks for any other available equipment among its station. After checking all equipment, if no lot is processable, it retries upon any state change in its station. These changes generally include: a busy/down equipment becomes available, a qualified operator becomes available, or a new lot arrives in the station's queue.

Once there are processable lots, the agent acquires the current system state s_t along with the reward r_{t-1} of the previous action a_{t-1} . Then, the state is fed to a DQN in which the evaluation network is used for selecting the dispatching action according to the most recent learnt policy. Each agent has its own DQN, hence they learn their policies individually. The available actions include the index of the lot to be dispatched, and no-dispatching action if the agent decides to wait before dispatching. The agent explores random actions according to an exploration probability. Accordingly, it gathers information and finds better policies, while also exploiting the current best policy to maximize rewards. The epsilon-greedy algorithm is a common approach to exploration, balancing the selection of the best action with trying out random actions to discover potentially better policies [94]. This is important for allowing the agent explore different policies and avoid performance plateaus.

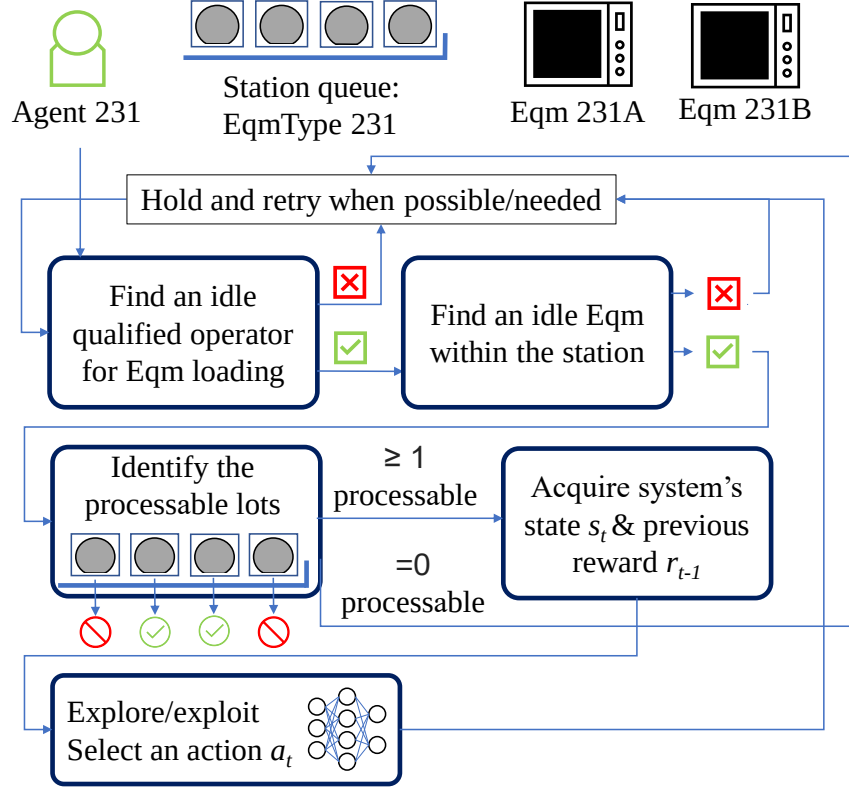


Figure 6.5 Dispatching agents through the simulation model

System state definition

The state definition plays a major role in how the agent perceives its environment, which is the simulated system in this case. The agent selects the action for each state given the learnt policy. This policy is updated and corrected according to the past states, selected actions, and rewards. If the state does not include the system's key information, the agent will not be able to learn the proper policy that leads to suitable actions. We define the state to include information for the waiting lots for each agent's station queue. It also includes global details for the other stations in the system. Our state definition for the waiting lots in the agent's queue are as follows:

- Timer state: within a timer, beginning of a timer, or not within a timer
- Timer limit: remaining time if within a timer
- Lot type
- Product ID

- Slack ratio (SR)
- Number of wafers
- Cycle Time X factor (CTX): total elapsed time ratio to the total value-added time

Agents need to get information about the rest of their system as well. For this, the state includes the following information:

- Number of waiting lots in each station
- Equipment state of all stations: up/down
- Number of lots arriving after (1-5) steps for each type of product

This state definition is done considering the collaborative behavior. Agents learn to work together in order to reach both their local goal, related to their station, and their global goal, which is related to the system as a whole. For example, agents at which timers start should learn when to start a timer. They also learn whether or not the subsequent steps' agents will be able to process that lot before exceeding the timer's limit. Similarly, the subsequent agents learn how to select the lots that are within timers. Their policies avoid exceeding the timer's limit and its consequences. Such consequences include lot junking or reworking which congests the system through the reworking steps. Accordingly, each agent is fed with representative information about its station and the rest of the stations in the system.

Our state definition permits scaling the number of stations up by adding just 1 neuron for each new station. This neuron accounts for the number of lots in queue of the new station and its state. To dispatch lots to the new station, a new agent with similar network size should be added for each new station. Finally, adding a new product is considered using the same agents as the product ID is passed as a number in the state definition. Hence, adding new products does not affect the network size or the training time.

Reward function definition

The reward function represents the feedback of the system for how adequate are the agents' actions. The complex nature of semiconductor production systems implies the importance of a well formulated reward function. This becomes more important in case of actions that have high impact on performance, such those related to timers processing. The later kind of rewards are assigned directly to more than one agent at once; upon timer limit's exceeding for example. In this case, assigning a single negative value of reward is not practical and

results in slow learning agents, or selection of trivial actions such as not starting any lot within a timer. Generally, the reward function in such complex systems highly affects the learning quality, hence it should be formulated carefully.

Similar to the state definition, our reward function include feedback that is related to the both agent's station and the system globally. Our reward function, as shown by Eqn. 6.3, includes the following:

- The general CTX for all products upon completion of production
- The average waiting time of lots in the agent's queue
- The average number of lots in the agent's queue
- The utilization level of the agent station's equipment
- The timers processing rating; representing how much this agent is capable of processing lots within timers
- The negative reward for exceeding timers which is assigned upon failing to respect a timer's limit

$$\begin{aligned}
 r_t = & \frac{CTX_{base}}{CTX_{MA}} + \frac{TQ_{wait,base}}{TQ_{wait,MA}} + \frac{NQ_{wait,base}}{NQ_{wait,MA}} + Util\%_{MA} \\
 & + \frac{TimerRating\%_{base}}{TimerRating\%} - TimerReward_{exceeding}
 \end{aligned} \tag{6.3}$$

In Eqn. 6.3, the components of our reward function are added to form the reward at time t . Agents aim to increase the value of their reward as much as possible. The reward function consists of many terms that are not necessarily similar in range. Therefore, we use base values, denoted by the subscript *base*, for some components to normalize these terms as possible with respect to each other. Also, we use a moving average for some components to provide a clear representation for the current trends, instead of absolute or total average values. We used moving average with constant window size to capture the performance indicators over time. This provides a relevant feedback for the agent's action after long operation time. For example, the CTX reward component is only affected by the relatively recent system's performance. The moving average is denoted by the subscript *MA*. The specific parameters, like moving average span and base values, are determined through experimentation and training runs. The timer reward is also customized to ensure that agents that exceed the

timer limit are penalized according to their share in this event. The reward is formulated through experimentation to find the most effective strategy.

The first term, CTX_{MA} , represents the general CTX of all finished products. For the CTX_{MA} , the lower the better, so that it is assigned as the denominator. The second and the third terms, $TQ_{wait,MA}$ and $NQ_{wait,MA}$, represents the moving average for the waiting time and number of lots in the agents' queue. They are assigned as the denominator for the same reason as the CTX_{MA} . The fourth term represents the moving average for the utilization, $Util\%_{MA}$, of the agent's equipment.

The last two terms, $TimerRating\%$ and $TimerReward_{exceeding}$, are related to agents which process lots within timers. The timer rating represents the ratio of total number of wafers whose timer limits were exceeded to the total number of wafers which were processed within a timer. A value of 5% for a certain agent means that this agent failed 5% of the times and resulted in exceeding their limits. The lower this value the better. The $TimerReward_{exceeding}$ is a value which is assigned directly when a timer limit is exceeded. If the timer involves 4 steps at agents A,B,C, and D's stations, and the timer limit was exceeded while the lot was at the agent C's step, the negative reward is assigned to each of A, B, and C. The value of the reward is not the same, and depends on the circumstances that might resulted in that situation.

There is a base value, $TimerReward_{exceeding,base}$, for the negative reward that is assigned upon exceeding a timer. Upon starting a lot within a timer, the risk of this timer is assessed according to how much the next timer's steps are congested. The more they are congested, the more risky is to start that lot. During the steps that are within the timer, B and C, the lot waiting time in each step along with the $TQ_{wait,MA}$ of that step's queue are stored upon starting processing in these step.

The value of $TimerReward_{exceeding}$ for each agent depends on the agent's order among the timers' steps. For the given example, there are 3 cases for getting the negative reward. Firstly, the timer starting agent's negative reward is proportional to the risk value. Secondly, agents which have already processed that lot get a negative reward proportional to the ratio of the lot's waiting time in their queues to their $TQ_{wait,MA}$. Finally, the agent which the timer was exceeded in its queue gets a negative reward similar to the previous agents and proportional to the ratio of remaining timer duration upon arriving in its queue to the timer duration. If the timer was exceeded in the queue of the timer's last step, the last agent's reward is doubled. However, if a timer was exceeded due to a failure of an equipment, no negative reward is assigned to any agent.

If the lot was a rework for a previously exceed timer, all the negative rewards are doubled

for all agents. This is because the timer's rework has only one chance of rework, otherwise it will be junked.

This gives the agents better feedback for their different actions regarding the lots that are within timers. This is more suitable than assigning a fixed negative value for all agents upon timer's limit exceeding. Such fixed reward value is not representative for each agents' action and results in bad learning.

6.5.5 DQN implementation

We used Arena simulation to build the simulation model as an environment to train the dispatching agents. Arena is capable of executing programming codes developed in Visual Basic for Applications (VBA) directly in the simulation model. This allows writing a code and communicating or automating other applications as well. We used this feature in Arena to pass the state and the reward for each agent to the learning algorithm. Also, it is used to receive the action and to pass it back to the simulation. We used Python 3.7 and TensorFlow library to handle the DQN training and execution. VBA works as the communication channel between the simulation mode and the agents' DQNs in Python.

6.6 Results and discussion

6.6.1 Case study description

Our simulation model is built for a real semiconductor manufacturing system as a part of a research project. The main objective of this project is to develop a decision-making support tool to address different production management-related problems. As mentioned in Section 6.4, the semiconductor industry includes many sources of production complexities. These complexities are problematic due to the high-mix of different products. In addition, the shifting towards a customer-centric production and its consequent induced variabilities and uncertainties.

We implemented a hybrid dispatching rule currently used in the real system. Also, we applied FIFO dispatching rule to compare a simple dispatching rule to the hybrid dispatching rule and the DQN dispatching.

The real system applies rule-based dispatching strategy. The hybrid dispatching rule is formed by combining the following four heuristics:

1. Earliest-timer's limit if any

2. Relative importance given to products
3. Slack ratio
4. Creation time

A priority is assigned for each lot according to the hybrid rule. For example, if all waiting lots have same or no timer's limit, the relative importance given to products is checked. Products' relative importance is assigned temporarily according to the current state of the production system. This includes lack of qualified operators for certain equipment types, maintenance in fabrication areas, and criticality of certain particular requests from a customer. Similarly, if the relative importance of the customer is the same, the Slack ratio is checked.

In some specific situations, the hybrid rule is bypassed. For equipment with capacity of more than one lot, lower priority lots may be dispatched with higher priority ones as long as they have the same recipe. The priority of a lot is subject to change due to the possible updates of the SR and/or the relative importance of customers.

Reworking a lot, whose timer's limit was exceeded, is not started unless all its timer's steps' equipment are held in advance. While an equipment is held, waiting lots are not dispatched even if they have higher priority. This way, the timer's rework becomes less probable to be junked due to limit exceeding. However, a timer's rework can only be done once for the same lot. In our case study, we simulated the system while applying the hold-in-advance for reworks. In addition, we tested the system without the hold-in-advance.

The hold-in-advance is applied for timers, at starting, whose limits are frequently exceeded, and which are considered risky. These risky timers are identified through the production history, and/or can be identified by the ratio between their total processing time and the max timer's delay. The timers risk is subject to alternations upon production modifications such as products alternations, process flows changes, and resources capabilities modifications. This means that timers which are considered less risky may become high risk upon any production-related alternation.

In this production system, it is non-optimal to rely on a fixed set of dispatching rules. The nature of the applied hybrid rule induces variabilities and uncertainties in the system. Additionally, continuously altering equipment's restrictions/dedications and sudden failures increase the variability. Optimization through DES is used in complex systems to find better dispatching solutions as discussed in Section 6.3. However, this is impractical as reconfigurations are continuously performed. Optimum solutions through DES are normally based on averaged output performance. Hence, there can not be a single set of rules that is optimal

all the time. Also, optimization techniques do not learn the interrelations among system's different objects such as resources, queues, and products at their different states.

The objective in this case study is to apply RL in DES as a decision-making support tool to explore its potential to replace the normally applied dispatching rules. Furthermore, to introduce an adaptive dispatching tool that is able to provide practical dispatching decisions in different challenging situations in which normal rules do not perform optimally.

We simulated two different products whose process flows passes by the same equipment in different steps through their production. The number of stations that these products pass by are 23 stations. Each station has its own agent. The simulation model starts empty. The model is populated using a simple order release through a probabilistic interarrival times for orders per product. For equipment state, the uptime/downtime is represented as stochastic time distributions.

The process flow for each product include some repetitive steps. Also, due to reworking, the same lot may repeat the same steps. The process flow of Product 1 includes a timer. This timer includes 3 steps which are performed in different fabrication areas. This timer's limit is a maximum of 10 hours. The timer processing time, excluding transferring and waiting time, is around 1 hour.

6.6.2 DQN configuration

The parameters related to the DQN are shown in Table 6.1. The input layer includes the information of up to 20 processable observed lots in the agent's queue, and the other information of the system as defined before in the state definition in Section 6.5.4. We observe the information of the waiting lots up to 20 due to computational limitations. The agent can select not to dispatch any lot and wait or dispatch one of the processable lots. For example, the agent may prefer to wait for an upcoming lot to fill its equipment capacity. The agents actions are passed through an action masking layer to prevent non-feasible actions such as selecting a non-existing lot. Table 6.2 shows the parameters for DQN training.

The DQN training is performed each 40 consecutive actions for an agent. A batch is randomly drawn from replay memory $\mathcal{D}$ for training. A copy of this network is used to replace the target network every 320 consecutive actions for an agent.

6.6.3 Results and comparison

To test the performance of the DQN dispatching agents versus the heuristic rule-based dispatching, we execute the simulation in both cases for a certain period of time. In addition,

Table 6.1 DQN's parameters for dispatching agents

Parameter	Value
Input layer size	176
Output layer size	21
Number of hidden layers	2
Hidden layer 1	140
Hidden layer 2	88
Learning rate	0.005
Discount factor	0.85
Activation function	ReLU
Optimizer	Adam

Table 6.2 Learning parameters for dispatching agents

Parameter	Value
Replay memory size	20000
Learning batch size	1000
Learning iteration	40
Target network replacement iteration	320

we applied FIFO dispatching rule, with and without hold-in-advance, to compare a simple dispatching rule to the hybrid dispatching rule and the DQN dispatching. The system starts with no WIP lots or no down equipment. The DQN dispatching performance is captured during the training of the DQN agents. The agents starts with random policies that are improved through the training and interaction within the simulation.

Figures 6.6 and 6.7 show the CTX in the system using the hybrid dispatching heuristics and using DQN dispatching respectively. The CTX for a lot is calculated before leaving the system for each product individually. Moving average of size 50 is used to show the trend in CTX results as a smoothed curve. The non-value added times are increasing over time for both products when heuristics-based dispatching is applied. This means that the level of congestion in the system is increasing. In contrast, the system using DQN dispatching holds non-value added times at lower levels.

Figures 6.8 and 6.9 show the percentage of rework and junk lots that are processed within a timer in both systems using hybrid dispatching heuristics and DQN dispatching respectively. The percentages are calculated with respect to the total number of lots that entered the sys-

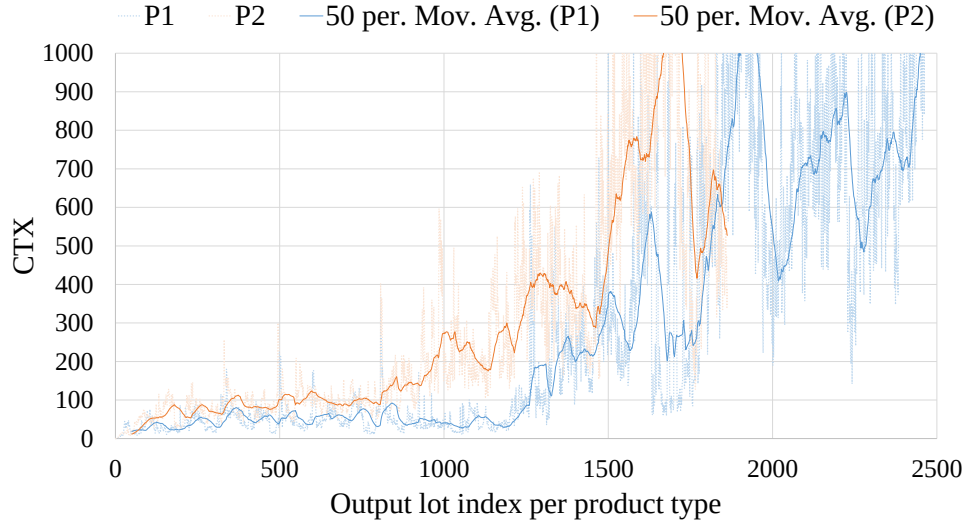


Figure 6.6 CTX in system using heuristics

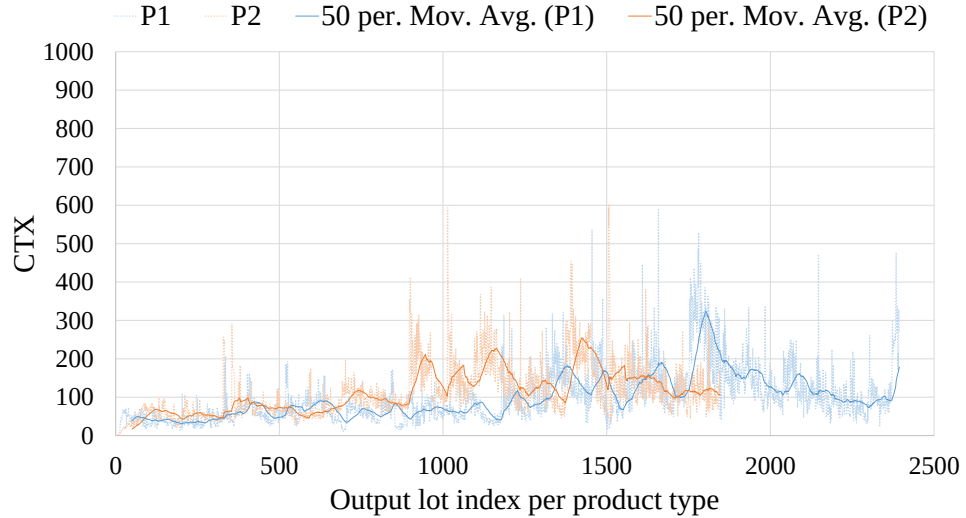


Figure 6.7 CTX in system using DQN

tem. Only lots that enter the system are counted without including those that are scrapped. Hence, the percentages of scrapped and reworked lots are calculated in relation to all lots through dividing the count of reworked and scrapped lots by the count of lots that entered the system. The system using heuristics-based dispatching is allowed to hold the equipment in advance upon having a rework for an exceeded timer, as applied in the real system. This means that a timer rework is handled as a critical timer that requires holding, seizing, all equipment through its timer's steps until it is processed through them. However, the hold-in-advance is not permitted for the system using DQN dispatching. Agents learn how to handle lots that are within timers. Figure 6.10 shows the rework and junk percentages in

system using the hybrid dispatching heuristics without the hold-in-advance. The percentage of lots exceeding timers is high starting from early stages of the simulation. In addition, the percentage of junk lots is higher. Figure 6.11 shows the rework and junk percentages in system using FIFO. Both rework and junk percentages are the highest compared to the other dispatching strategies. The figures show that agents applying DQN dispatching perform the best in handling timer processing.

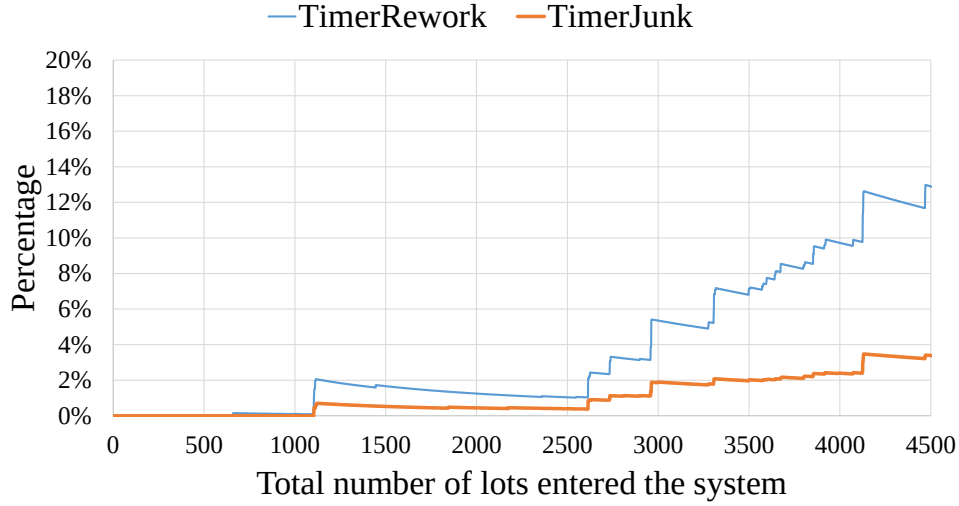


Figure 6.8 Timers rework% and junk% in system using heuristics

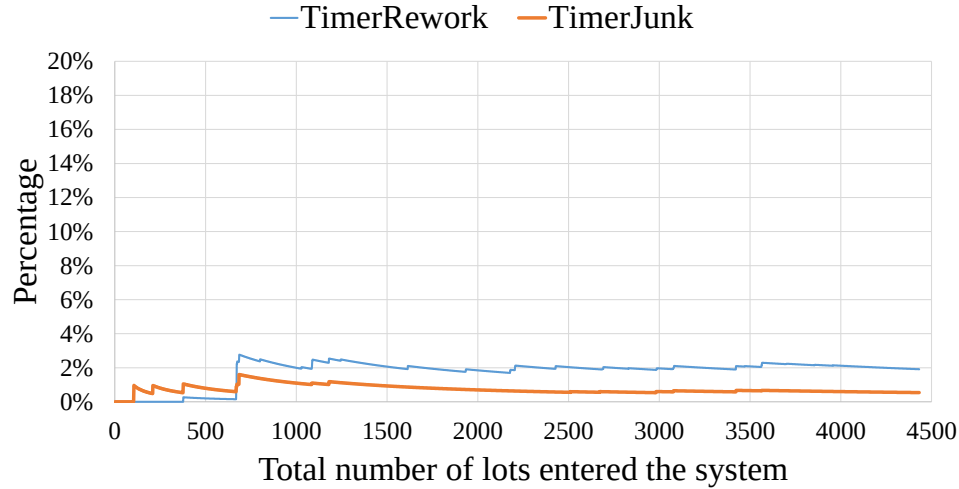


Figure 6.9 Timers rework% and junk% in system using DQN

Figures 6.12, 6.13 and 6.14 show the throughput of the system for each product. The system using DQN dispatching achieves higher and more steady throughput than the system using heuristics-based dispatching with or without hold-in-advance. This is due to the less

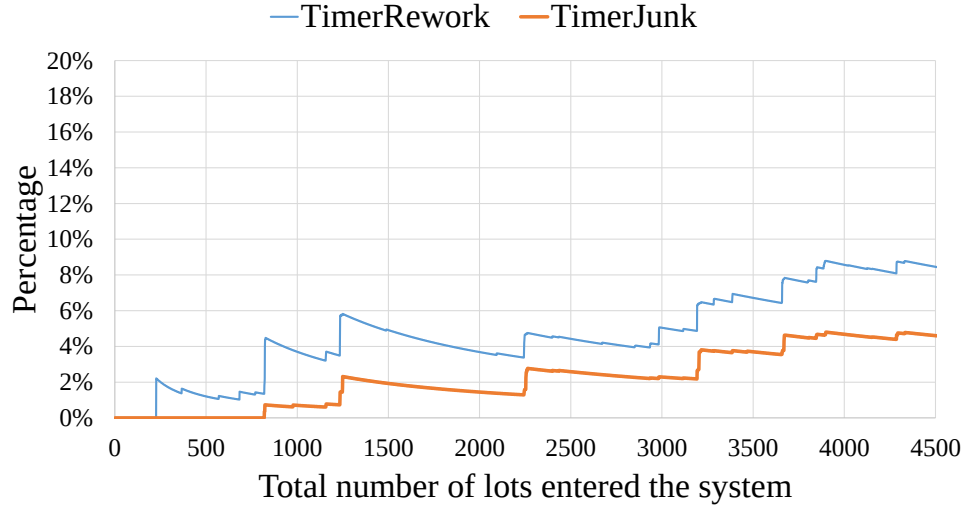


Figure 6.10 Timers rework% and junk% in system without hold-in-advance

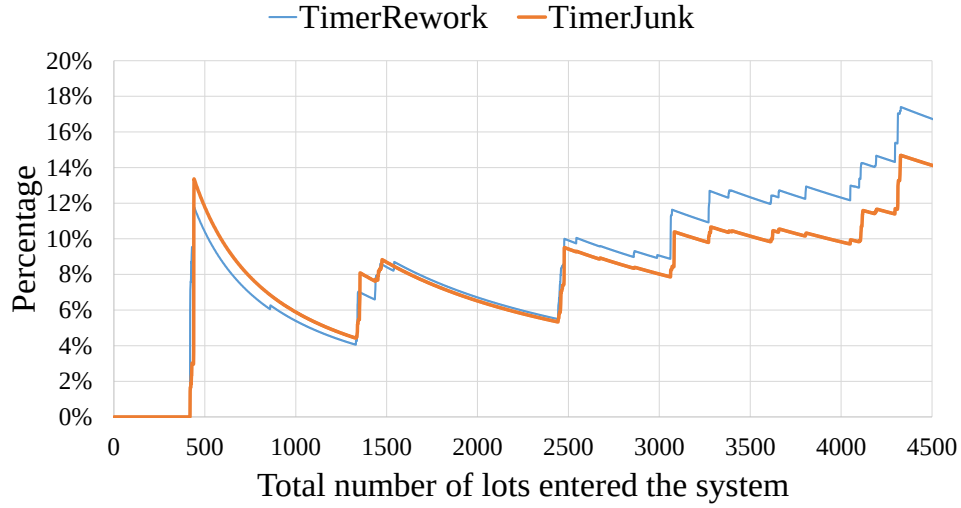


Figure 6.11 Timers rework% and junk% in system using FIFO only

non-value added times that is achieved by the DQN agents. In addition, the timers' rework and junk percentages are lower for the system using DQN. As shown in the figures, the hold-in-advance increases the level of congestion in the reworking steps to decrease the junk percentage. This results in higher non-value added times, hence a lower throughput. Figure 6.15 shows the throughput of the system using FIFO which has the lowest throughput compared to the other dispatching strategies.

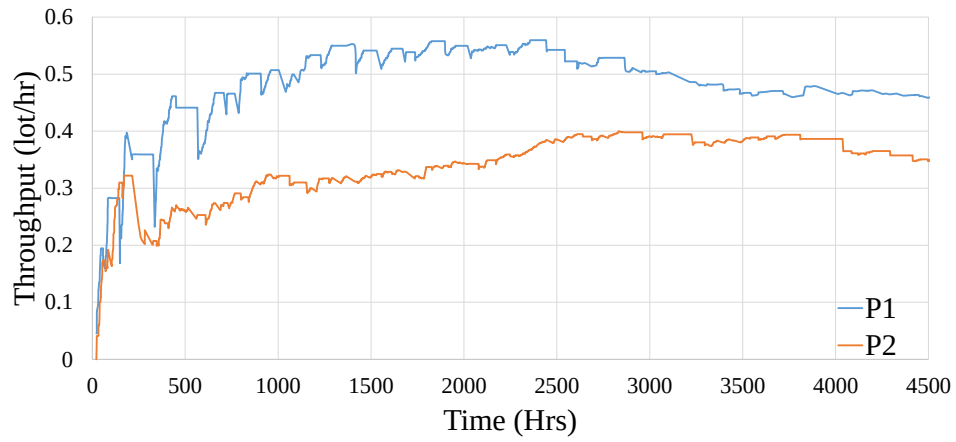


Figure 6.12 Throughput of system using heuristics

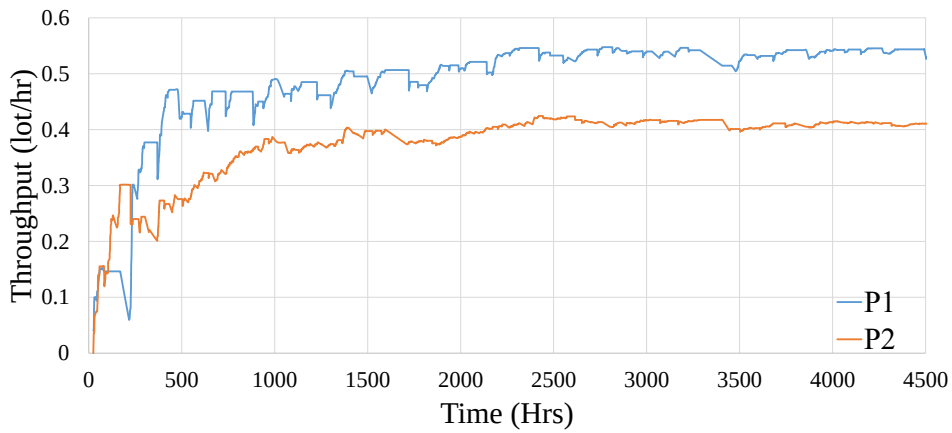


Figure 6.13 Throughput of system using DQN

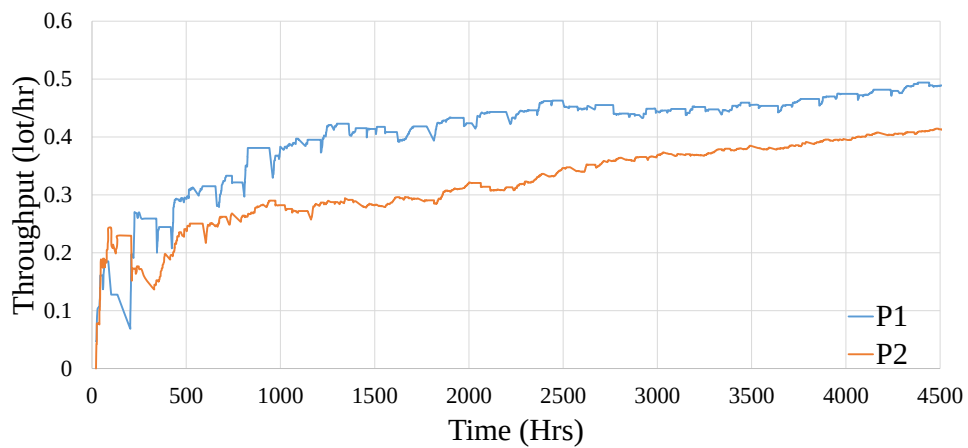


Figure 6.14 Throughput of system using heuristics without hold-in-advance

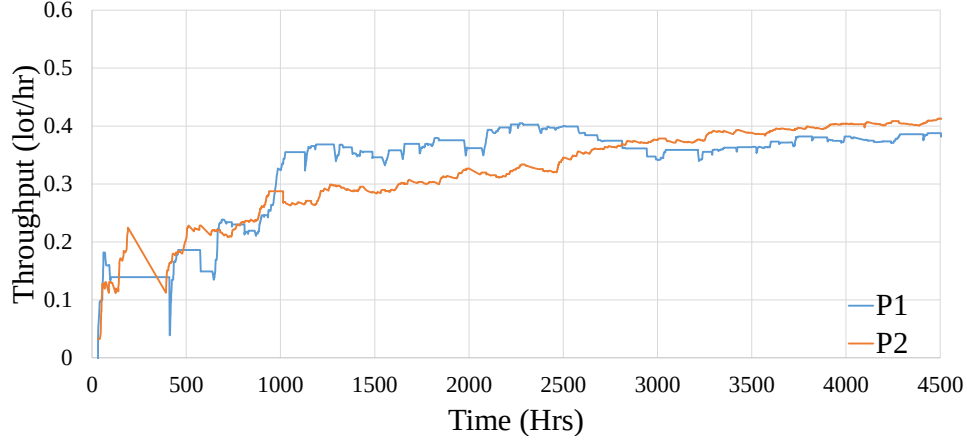


Figure 6.15 Throughput of system using FIFO only

6.6.4 Discussion for DQN agents' performance

As we mentioned in Section 6.5, each agent is trained individually. The reward includes performance feedback related to the agent's station, and the whole system. From the performance shown in the previous figures, Fig. 6.7, 6.9 and 6.13, the agents individually developed proper dispatching strategy that improves their station's performance and the general system's performance.

In addition, the agents learnt cooperative behavior to handle challenging situations. As an example for such cooperation, 3 different agents learnt to timely coordinate the processing of product 1 within its timer. This timer has the following steps:

1. Station A processes the first step at which the timer starts. It has a single equipment with a single lot capacity. The processing time takes around 6 minutes.
2. Station B processes the second step. It has 2 equipment with capacity of one lot for each. Product 1 has a restriction for one of the equipment while product 2 can be processed in both. The processing time for product 1 is around 47 minutes.
3. Station C processes the final step. It has a single equipment with capacity of 2 lots at a time. The processing time is around 13 minutes.

The decision of starting that timer is considered critical due to the short processing time of station A compared to the subsequent steps. Agent A will congest station B with timer lots, if it decides to start many timers sequentially. At the beginning of learning, Agent A was making improper decisions by starting many timers back to back. This resulted in having

long waiting in station B and many timers' limit exceeding as shown in Fig. 6.9. Later, Agent A learnt how to cooperate with the subsequent stations' agents. Moreover, Agent B learnt to process product 2 more often in the equipment which has restriction against product 1. This behavior makes the other equipment more available for product 1 timer lots. Agent C learnt to wait until filling its capacity, as long as this will not cause a timer to be exceeded or longer waiting time.

The training of all our agents took around 75 minutes on a 2.7 GHz core i7 CPU and 8GB of ram. This includes the time consumed for the simulation on Arena and the communication between Arena and Python through VBA. It takes around 0.3 msec for an agent to make a single decision.

6.7 Conclusion

In this paper, we presented a DQN-based dispatching and resources allocation approach for a complex production system. The studied complex system is a case study for a semiconductor manufacturing system. The system is simulated through a data-driven agent-based DES using Arena simulation. The simulation provides the training environment for our DQNs agents. The agents are responsible for dispatching the products and allocating equipment through the execution of the simulation. Each agent manages a single station of one or more equipment. When a dispatching action is needed, the simulation provides the agents with the system's state. The agent then selects the proper action given that state. The system then applies the action and rewards the agent. The reward corresponds to the station's performance and the system as a whole.

Our DQN approach is compared to the heuristics-based dispatching that is currently applied in the real system. The DQN agents learnt, through the simulation, to improve stations' and system's performance. The agents learnt to cooperate in challenging situations such as timers' processes. Our results show an improvement in the performance using DQN agents. This improvement is shown as a general increase in the throughput of the system, reduction in the total non-value added time percentage, and reduction of the timers' exceeding and hence the consequent junk.

Our results are achieved through the proper definition of the system's state information and the reward function. This is crucial for complex dispatching decisions such as these related to timers processes. The reward function should be carefully defined to enable the cooperation among agents. A well formulated reward function prevents the agents from falling into trivial decisions.

Acknowledgement

This work is funded through the Mitacs Accelerate program, Grant No. IT01632, in partnership with Teledyne DALSA Semiconductor, a business unit of Teledyne Digital Imaging, Inc. The authors would like to thank the industrial and information technology teams at Teledyne DALSA semiconductor for their great support and collaboration.

**CHAPTER 7 ARTICLE 4: DEEP REINFORCEMENT LEARNING FOR
ADAPTIVE DISPATCHING AND ORDER RELEASE CONSIDERING
CONFLICTING HUMAN AND EQUIPMENT INTERRUPTIONS**

Ahmed H. Sakr, Ayman Aboelhassan, Soumaya Yacout, Samuel Bassetto

Submitted to:

Engineering Applications of Artificial Intelligence

December 28, 2022

7.1 Abstract

In reinforcement learning, the adequacy of decision-making relies on the difficulty of the training environment. In a production management context, training environments are achieved through simulation, imposing a certain level of abstraction of the reality. Traditionally, challenging events are simulated probabilistically to account for the variability in real systems. Such events do not necessarily impact the performance of the system all the time. Hence, reinforcement learning algorithms need prolonged training to experience more events and learn to overcome their consequences. However, performance improvement of reinforcement learning through extending training time is not a practical solution.

In this paper, we improve the learnt policies for decision-making of managing agents, dispatchers and order releasers, through the concept of self-play. We introduce conflicting agents simulating real interruptions that occur to system resources. They apply reinforcement learning for selecting the resources to interrupt. This improves the learnt policies without prolonging training time. As for a production system, this accounts for real systems variabilities increasing the readiness of such policies to be applied in reality. We employ an agent-based discrete-event simulation as the training environment. This is presented through a use case for a real semiconductor manufacturing system. The conflicting agents learn to interrupt operators and equipment that significantly affect system's performance. They actively force the managing agents to explore new strategies for dispatching and order release to tackle the interruptions impact on the productivity of the system.

Our approach is compared to heuristics-based strategies, that are applied in our use case, and to a system with managing agents without interruptions. Our approach achieved better production performance than the currently applied strategy. In addition, within the same training time, the conflicting agents helped the managing agents to learn better policies than policies learnt without system interruptions.

Keywords

Deep reinforcement learning, Discrete-event simulation, Agent-based modeling, Conflicting agents, Human-AI symbiosis

7.2 Introduction

The new industrial revolution, known as Industry 4.0 (I4.0), has initiated a significant employment of artificial intelligence and new technologies in production systems [95]. These

technologies are based on digital connectivity among system's components including system resources, machines and operators, and products. This facilitates data collection, information sharing, and knowledge extraction [96]. I4.0 technologies include Cyber-Physical System (CPS), Digital Twins (DTs), and the connectivity channels provided by the Internet of Things (IoT). Through CPS, powerful and rapid decision-making are provided for productivity improvement [97]. Within a CPS, DT provides virtual representation of the physical system and applies its decision-making support tools including artificial intelligence [98]. Decision-making decentralization and autonomy, along with integration of human aspects, are among the challenges for production management within the I4.0 paradigm [99]. However, they have been identified as the strategic responses of manufacturing given the rising market competitiveness [2, 3].

Decision-making requires identification of intercorrelations and interdependencies which is difficult for analysts [24]. In production management, dispatching and order release are among the key decisions for productivity and waste reduction. Traditionally, they are managed through heuristic-based rules. As DTs provide artificial intelligence for decision-making, production systems benefit from algorithms such as Reinforcement Learning (RL) to replace heuristics. These algorithms train decision-making policies. The policies are initiated with random parameters. They are improved through interaction with the training environment where they are applied. Training environments provide feedback so that the algorithms update and accordingly improve their policies.

In a production management setting, off-line training is performed through simulated environments. This is a practical methodology, as compared to on-line training, as RL training requires hundreds of thousands of decision-making episodes [100]. After training, the learnt policies are applied in the corresponding real system. Their performance in the real system depends on the abstraction level of the simulated environment. In addition, the performance in different challenging situations relies on whether similar events were simulated through the training.

Through simulation, some challenging events are normally introduced on random basis. Challenging events involve interruption of system resources such as human operators and equipment. This may introduce some challenges for the policies that are trained to improve their readiness for similar situations in reality. Due to being random, these events do not consistently affect the general performance. This requires long training time for the policies to experience these occurrences and learn to overcome their effects. However, long training does not necessarily yield better learning.

Within CPSs, there are efforts towards integrating human operators instead of decreasing

their presence [101]. Human integration has started to draw attention for enterprises. This is due to the smooth realistic transition it provides for employing I4.0 technologies [102]. Human operators, as resources, are not always available when idle. They make/need some work interruptions. In this paper, we apply RL for a model-based decision-making. It is applied for a semiconductor manufacturing system through a real use case. The application is based on a hybrid simulation of Discrete-Event Simulation (DES) and agent-based simulation, as a virtual training environment. The agents are employed for decision-making within the model. Their decisions are provided through RL algorithms.

Our contributions include providing a comprehensive approach for prompting learning algorithms to improve their policies. This is done through the introduction of RL-based conflicting agents applying the concept of self-play. Our conflicting agents present the variability that resources' interruptions, operators and equipment, causes in such systems. This develops better decision-making policies than training with no conflicting agents, and without extending training periods. The RL-based conflicting agents existence improves the managing agents readiness to be deployed in real systems, and to work in environments where human operators are present.

In our RL setting, agents make decisions through Deep-Q-Network (DQN) algorithm. Managing agents are responsible for order release, and dispatching and resources allocation applying the concept of decision making decentralization. For our use case, there is an order release agent that decides when to start next lot for each product type. Similarly, there is an agent for each equipment station that makes decisions for dispatching and equipment allocation.

For the conflicting agents, they are responsible for interrupting system's resources. There are two agents, one for interrupting the operators and the other is for interrupting the equipment while processing. In the RL setting, they learn to interrupt operators and equipment that significantly impact the system's performance.

The managing agents are trained to improve the system's performance. This includes local objectives on resources' level, and global objectives on system's level. Conflicting agents learn to impact the system's productivity. This reduces the chances that their interruptions may be ineffective as in case of random interruptions. Conflicting agents actively force the managing agents to explore new strategies for tackling their impact on the system's performance. Also, this accounts for the existence of human operators. Our environment's state formulation considers these manufacturing systems' continuous changes such as process flow updates/modifications. This minimizes the effect of these changes on the learnt policies. Our reward functions formulation promotes the cooperation among managing agents.

The rest of the paper is organized as follows: Section 7.3 presents a literature review for

production management problems in complex production systems. It also presents the available solutions through simulation and reinforcement learning. Section 7.4 states the problem in the semiconductor industry as an example for a complex production system. Section 7.5 presents our methodology for employing the RL for both managing and conflicting agents. Section 7.6 presents a semiconductor manufacturing system use case. It presents our results compared to the currently applied heuristics-based strategy. Finally, Section 7.7 provides a conclusion for our works and results.

7.3 Literature review

7.3.1 Reinforcement learning for decision making

Decision-making within production systems is normally performed through the traditional approaches such as analytical methods and precise simulations. These approaches provide solutions that become infeasible overtime due to the dynamics and the uncertainties of production systems [103]. Adaptive management strategies, such as Machine Learning (ML) algorithms, are used to tackle these challenges, especially in complex production systems. ML has the capacity to handle decisions related to production management through learning strategies in a data-driven scheme [104]. Industrial applications of ML show several benefits such as productivity improvement and cost minimization [105]. This shows the promising profits that can be yielded through the synergism of ML strategies and production management.

Within ML, RL techniques involve agents that interact with their environment. Generally, an agent is a software component that perceives, acts, and communicates to perform certain tasks [20]. Agents provide different properties such as autonomy and responsiveness. In RL, agents learn to take actions that maximize their reward given their environment's state. The reward represents the feedback of the environment [21]. It represents the adequacy of the taken actions. Accordingly, agents are trained to improve their action selection policies. RL is suitable for problems where there is no known supervising solution. Hence, agents explore different actions, learn from their experiences, and exploit their learnt policies.

Paraschos et al. [106] presented a RL-based approach to provide a joint production, maintenance, and product quality control policies. They applied their approach for a stochastic production/inventory system that is encountering several failures. RL-based approach was selected due to system complexity in which methods such as dynamic programming are not appropriate. This was done to find a trade-off among contradicting performance metrics. Their RL approach showed promising results for application in real-world systems. Xanthopoulos

et al. [107] proposed an adaptive CONstant Work in Process (CONWIP)-type control policy based on RL. They applied their approach to a stochastic manufacturing system including a number of series work stations. Their RL agent made decisions of capturing/releasing production authorizations. The aim was to minimize the costs of holding and backordering. Their approach was shown to outperform the alternative control policies. Stricker et al. [60] implemented RL-based adaptive control for order dispatching in semiconductor industry. They tested their approach for order dispatching among 3 work stations. The implemented approach was compared to traditionally applied rule-based heuristics. Their results showed an improved performance as compared to a system applying heuristics. Liu et al. [66] used an actor-critic model for RL for solving job shop scheduling problem. They trained multi-agents with a common neural network in a parallel training scheme. Their agents select different dispatching rules at each dispatching decision. Their results show a good balance between makespan and execution time in dynamic environments.

Although RL is considered in the literature for production management, the research domains are still barely explored or not addressed. Some possible reasons includes the rare consideration of human aspects when linking ML generally to these problems [104]. Also, real-world challenges are rarely considered when applying these techniques.

7.3.2 Human operators in Industry 4.0

Industry 4.0 has emerged with many possibilities for enhancing systems' flexibility and efficiency through the IoT technologies. These evolving technologies based the I4.0 on CPSs. Within a CPS, communication and computational capabilities are integrated with physical processes for improving the physical system's performance.

Automation has succeeded to replace several tasks that used to be performed by humans. However, conventional automation can not handle the increasing flexibility and adaptability requirements [101]. In addition, human presence is still indispensable given the increasing level of production demanding requirements [102]. The computational capabilities of CPSs have shifted the focus from the physical tasks to cognitive ones. This includes reasoning, learning, managing, and planning to reach human integration and symbiosis within CPSs. Accordingly, these systems become a course for human operators to continue working within these systems instead of being expelled [108].

Given the role of human operators and their important presence in production systems, the intelligence provided for decision-making should consider their presence. RL agents need to be trained in an environment in which operators are simulated. This includes the additional variabilities that humans typically cause, and the non-optimal/conflicting actions that they

may take. This enriches the policies that the agents learn through experiencing realistic situations and challenges.

7.3.3 Conflicting RL for improving learnt policies

RL has been implemented in the field of computer games to create artificial agents that play against human players. Computer games are considered abstraction of many real-world problems [109]. In such games, some artificial agents are required to cooperate or compete to reach their goals. This is similar to our system where managing agents learn to cooperate and improve the general system's performance, while conflicting agents learn to impact the performance. Many researches proved the effectiveness of the self-play concept, in which agents are operating against each other [110]. It forces agents to actively discover new strategies to outperform each other [111].

In this study, we broaden the preceding RL and DES application in our previous study [1]. This is done through a multi-agent environment in a complex production system with real challenges. Our managing agents learn to cooperate together to improve the system's performance considering various production requirements. The training environment is provided through a DES model for a real semiconductor manufacturing system as a use case. We reach adequate policies for the agents through our formulation of environment's state information, and our comprehensive reward function. The reward function involves local and global performance measures that represents agent's actions adequacy including its goodness of cooperation with closely related agents.

We applied the conflicting agents concept for interruptions that happen to equipment and human operators. This is motivated by the fact that, within a CPS, decision-making artificial agents must consider the human operators presence along with the conflicts that they commonly cause. RL conflicting agents enrich the managing agents learnt policies through forcing them through affective interruptions. This is similar to how self-play improves reinforcement learning in games applications. The full details of our approach are presented in Section 7.5.

7.4 Problem statement

7.4.1 Reinforcement learning-based decision making

In production management applications, reinforcement learning is considered one of the promising approaches for decision-making support. Reinforcement Learning agents are trained

within a simulated environment in which they develop their decision-making policies. According to situations' difficulty, agents learn and adapt their policies to improve their reward. Readiness of trained policies for different real situations relies on the difficulty of training within simulation. Training difficulty refers to simulated challenging events that agents have to make decisions to avoid their undesirable consequences.

There are several challenges in production systems related to production processes and resources capacity. In simulation, there is a certain level of reality abstraction which implies limited reality representation. Normally, events such as resource interruption are not as frequent as events related to dispatching for instance. This limits the opportunities to experience these events and learn to overcome their effects. Extending simulation time results in more occurrences of interruption events so that agents experience them. However, it is not the most practical approach and it does not necessarily improve the policies. Significant less-frequent events slow down the learning convergence and add high variance [112].

7.4.2 Outlines of semiconductor industry

Semiconductor manufacturing is considered one of the most complex production systems. In this industry, integrated circuits are built through repeatedly building thin layers of silicon through processes consisting of a number of steps [113]. Each step is necessary to build a single layer on the semiconductor wafer. This cycle is repeated several times according to circuit's design to reach a functional circuit. The processes are done in clean-room environment through different separated fabrication areas. In these fabrication areas, various work stations are located in which similar equipment are gathered. A product is normally processed through several fabrication areas according to its process flow.

Wafers are transferred in groups, usually 25 wafers, of a certain size. These groups are called lots and they are transferred in cassettes through processing steps. Equipment have different processing schemes for lots. Some equipment process one wafer at a time and require operator to feed them with wafers one by one. Other equipment can be loaded by a whole lot, or certain batch of wafers, at once and do not require operator's involvement during processing. Other equipment are capable of processing more than one lot at once. Some equipment are clustered in a single frame which is equipped with a robotic manipulator arm. This manipulator is responsible for loading/unloading these equipment with lots/wafers according to their processing scheme. These clustered equipment have loading choppers through which operators load/unload the lots.

According to each operator's qualifications and training, they operate certain equipment types. While lots are being processed within the equipment, the process may be interrupted

due to a warning for example. Thus, the responsible operator needs to check the equipment so that the held process can be resumed. Operators are also subject to interruptions during their shifts. These interruptions emerge due to the need for supervision in case of specific situations, usually related to quality issues. This varies the availability of operators and processing times accordingly.

Processing steps rely on recipes that are shared among many process flow steps for several products. This is similar to process manufacturing scheme. This industry also has the characteristics of discrete-manufacturing such as intersecting process sequences. This involves logistic challenges such as re-entrant flows due to repetitive processing cycles and varying equipment availability. These complexities have high impact when there is a high-mix of products. This implies make-to-order or engineer-to-order strategy which adds more pressure on system's logistics due to the increased level of variability [90].

7.4.3 Within products' process flow

Through the process flow, some successive steps form a timer with a certain time limit. In this case, the lot should reach the last step within the timer once it starts the timer's first step. If the timer's limit is exceeded, the lot either have to be reworked, starting from the timer's first step, or be scrapped in some cases. Scrapping a lot means losing all the added value of the preceding steps. Managing timers is normally difficult given the intersecting flows of different products sharing same equipment.

Lots may need splits. Generally, some steps need to be reworked due to quality reasons. Hence, the lot is split into two parts. The original part continues processing until a certain rejoin step. Through processing, it may have other parts that need rework as well. At the rejoin step, it waits for its other part(s) that is(are) reworked, then it continues through the subsequent steps. Similarly, there are IF-condition steps at which certain quality-related parameters are measured. Each IF-condition step has its *Then* and/or *Else* steps. According to the measured parameters, the lot is split and processed. Splits are rejoined after the IF-condition processes are done for each split. In many cases, IF-conditions are nested, and split parts may themselves be split through the nested conditions.

7.4.4 Management challenges

Semiconductor manufacturers have many Key Performance Indicators (KPIs) that determine the adequacy of the applied management strategies. Generally, these KPIs cover lot's cycle times, lead times, throughput, and waste percentage. There are other indicators that measure

job orders tardiness. Performance measures are important for decision-makers to understand their system, and realize its capacity. They are also essential for decisions such as due dates setting for new orders.

Dispatching and order release, or lot start as referred in this industry, have significant impacts on the system's performance. Traditionally, dispatching is managed through heuristic rules such as First-In-First-Out (FIFO), Shortest Processing Time (SPT), and Earliest Due-date (EDD). As job orders consists of a number of lots, the EDD is usually replaced by Slack Ratio (SR). The SR is the ratio of the remaining time till the due date to the remaining quantity to be produced to fulfill the order requirements. In case of presence of lots within timers, they do not follow the normal dispatching rules. Rather, they are dispatched based on how close they are to their timer's limit.

For order release, there are two types of traditional release policies, Open-loop Release Policies (OLRP) and Closed-loop Release Policies (CLRP). OLRP includes constant and random interval between successive lot starts. CLRP considers the state of the system and relies on some thresholds for lot starts. Examples for CLRP are CONWIP and Workload Regulation (WR). Normally, CLRP performs better than OLRP. But when productions systems are highly loaded, all policies perform similarly [114].

Process flow encounters many stochastic events, hence it becomes dynamic. These events depend on the system's state including Work-In-Process (WIP), system's resources availability, and job orders' due dates. For dispatching and order release rules or policies, there is no single solution that performs the best in all situations. Given these aspects, it is challenging to manage such systems and keep adequate level of performance with the traditional techniques. In systems with high dynamics, policies that learn and refer to the system's state are considered possible solutions to provide the required flexibility and adaptability.

7.5 Methodology

In our methodology, we employ DES for manufacturing system modeling. DES is selected for its capacity for capturing systems dynamics. DES offers suitable training environments for agents applying RL. We apply agent-based modeling scheme to provide the required functionalities for agents within the model. In RL context, agents functionalities are perception of their environment, and application of their selected actions. The environment provides feedback, or reward, to agents so that they fulfill their training loop. Details are presented in the following subsections.

7.5.1 Model building through DES

RL agents need a virtual environment, or a model, to be trained for making decisions. The quality and adequacy of agents' learnt policies depend primarily on the model's level of abstraction. The model creates realistic scenarios for agents, and provides reward for their selected actions. It activates the associated agent when a decision is needed, provides the required information about system's state, applies the agent's action or decision, and rewards the agent(s) according to its actions adequacy. Given the essential role of the model, as a training environment, we select a DES building technique that supports our system's behaviors and aspects. In addition, our selected technique adopts agents as decision-making entities. To achieve these building aspects, we merge data-driven and agent-based modeling techniques.

A data-driven modeling provides a compact modeling and supports representing complex behaviors and operational aspects such as semiconductor manufacturing ones. Hence, we provide various scenarios for the agents throughout their training within the model. In a data-driven DES model, the data drive products, or lots, within the model. Process flow, processing steps, and required resources' information are all acquired from the model's input data. Data-driven model also permit detailed applications for agents within the model. This is achieved through having modules for different functionalities including the ones that are required by agents to operate.

There are two types of agents in our model, managing agents and conflicting agents. The managing agents are order release agents and dispatching agents. An order release agent is assigned for each product type that is responsible for starting the lots. For dispatching agents, an individual agent is assigned for each equipment type that is responsible for dispatching waiting lots. The conflicting agents are an operator interrupter and an equipment interrupter.

There are two main setting for agents in the model: 1) implementing DQN as a learning and decision-making methodology, and 2) applying predefined logic in a preprogrammed scheme. In our simulation, there are two settings for managing and conflicting agents. In normal setting with no RL, managing agents apply their predefined heuristics, and conflicting agents make random selections. While in RL setting, managing and conflicting agents utilize DQN algorithm.

7.5.2 Data-driven modeling

In data-driven models, only system's operational behaviors are implemented in the model. Information related to lots' processing steps is acquired from model's input data. These data

include information that is required by agents. For instance, agents data include heuristic rules for dispatching and order release when not applying RL. When applying RL, agents data include state definition information.

For our system, the data are categorized as follows, as shown in Fig. 7.1:

1. Resources
2. Processes
3. Products
4. Agents

In Fig. 7.1, different data groups are linked to each other within the model's input data. This is shown by having similar colored icons. Generally, there are resources, such as equipment and operators. Resources provide processes for products according to their process flow.

In our model, resources data are for equipment and operators. For equipment, data involve equipment type and ID, loading scheme, capacity, and uptime/downtime distributions. For operators, working shifts and qualification for each operator are provided. Qualifications define which equipment operators are trained for. Operators availability information defines different interruption situations that operators may randomly have, when interrupted, including duration and probability.

As recipes are shared among products, processes' data include all recipes that are available within the system including their associated details. Recipes' details determine their processing equipment type, time, and yield percentages. The recipe type determines whether it needs specific pre/post processing or not. Among processes data, there are timers, rework, and IF-conditions including their details that are shared with products' process flows.

Products have process flow in which processing steps, through recipes, are determined according to the product design. Each product has its details for rework and IF-conditions such as probabilities of having a rework or not, how many wafers on average need rework, and same for IF-conditions. Also, job orders details are part of products' data. This provides orders quantities and due dates.

Agents data are defined also among the input data of our model. In normal simulation setting that does not include RL, order release, dispatching, and conflicting agents have their default pre-programmed working logic requiring specific data. In RL setting, DQN agents' data are related to state/reward calculation.

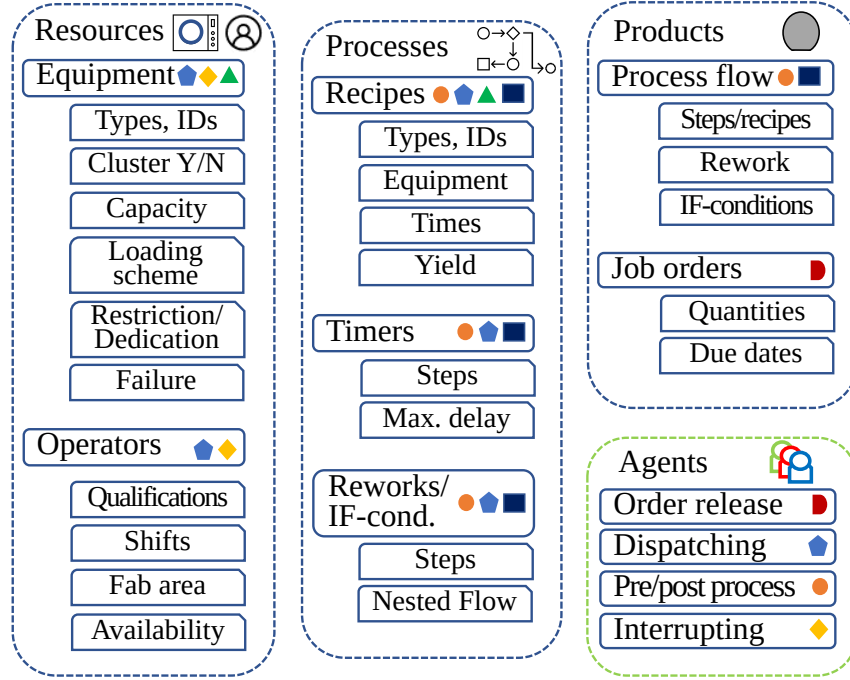


Figure 7.1 Data categories and relations of the proposed data-driven modeling for a semiconductor manufacturing system

7.5.3 Agent-based modeling

The main functionalities of the model are attained through agents. These agents operate within a number of sub-models. These sub-models handle tasks such as order release or lots start, dispatching waiting lots at each equipment station and allocating required resources, and other pre/postprocessing procedures. Figure 7.2 shows the different functionalities of agents that we employ in the model. In RL setting, our agents are distinguished as follows:

1. Pre-programmed agents which apply pre-defined policies for handling different pre/postprocessing tasks
2. Managing DQN agents which learn through RL to improve system's performance through making decisions for order release and dispatching tasks
3. Conflicting DQN agents which learn through RL to oppose managing agents through interrupting system resources

Pre-programmed agents handles pre/postprocessing tasks. These tasks include routing lots and their wafers according to the process flow of each product and considering possible

splits/rejoins due to reworks or IF-conditions. In addition, they handle timers starting procedures. Due to timers' limits, a timer is not started unless all equipment, that are needed within its steps, are functional. For normal setting with no RL, timers' criticality is determined based on relating their steps average processing time to the timer's limit. If it exceeds a certain pre-defined threshold, the equipment of processing steps are held in advance to ensure availability for that lot. This is performed also for lots which need timer's steps rework due to exceeding the time limit. The hold-in-advance is done regardless the criticality of the timer as timer rework is normally permitted once, if possible.

For managing agents applying DQN for their decisions, holding in advance for critical timers is not permitted. Agents have to learn to properly start timers, handle their steps, and avoid limits exceeding. Only equipment state is checked, before timers start, to ensure functionality.

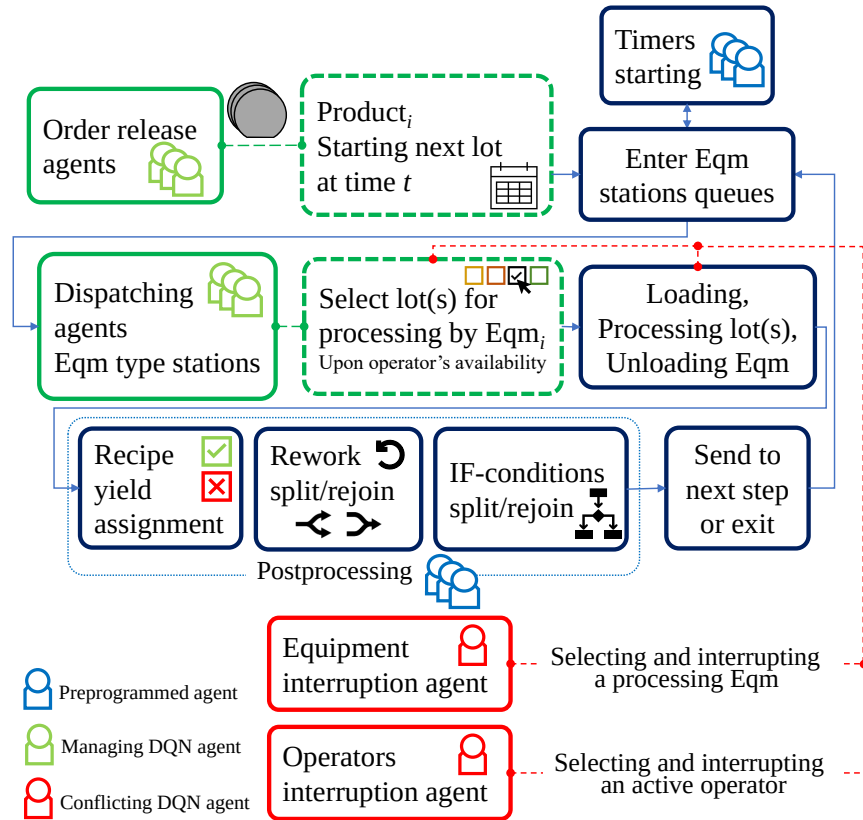


Figure 7.2 Agents through the proposed agent-based modeling for a semiconductor manufacturing system

In RL setting, managing and conflicting agents learn their decision-making policies through DQN algorithm for RL. Policies are learnt by agents through interacting with the simulation model as their training environment. Figure 7.3 shows how agents learn decision-making

policies through interaction with their training environment. When a decision is required, the environment passes the state s_t to the associated agent. The agent selects an action according to its policy given the current state. Then, the action a_t is passed to the environment to be applied. The environment sends its feedback or reward r_t to the agent. Along with the associated state and selected action, the DQN algorithm performs the learning process and updates the agent's policy accordingly.

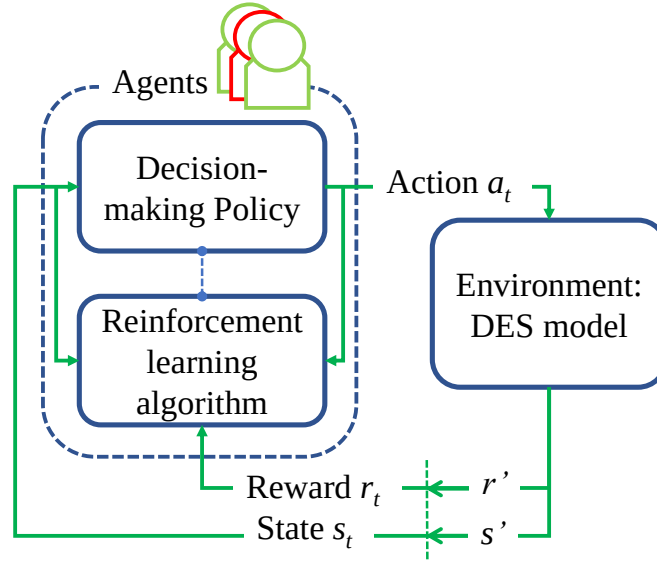


Figure 7.3 Agents Learning process through RL

7.5.4 DQN agents

Figure 7.4 shows how Deep-Q-Network agents' learning process and action selection. In this algorithm, artificial neural networks are utilized to predict the expected reward, Q-value, for each action a given the state s and following the policy π . Hence, networks perform Q-function approximation. The optimal Q-function, $Q^*(s|a)$, defines the maximum reward that can be found following the optimal policy. That function follows the Bellman optimality equation as [93]:

$$Q^*(s|a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s'|a')] \quad (7.1)$$

The discount factor γ , ranging in $[0, 1]$, discounts the future rewards as compared to the immediate one. As γ approaches zero, agents are shortsighted and consider immediate rewards only. When γ approaches 1, agents are farsighted in which delayed rewards are considered. This is helpful when actions' effects are normally delayed such as in production systems.

Agents adopt the ϵ -greedy technique for exploration/exploitation. ϵ value, ranging in $[0, 1]$,

represents probability of taking random action instead of applying the learnt policy. This is required to let the agent explore different unlearnt policies.

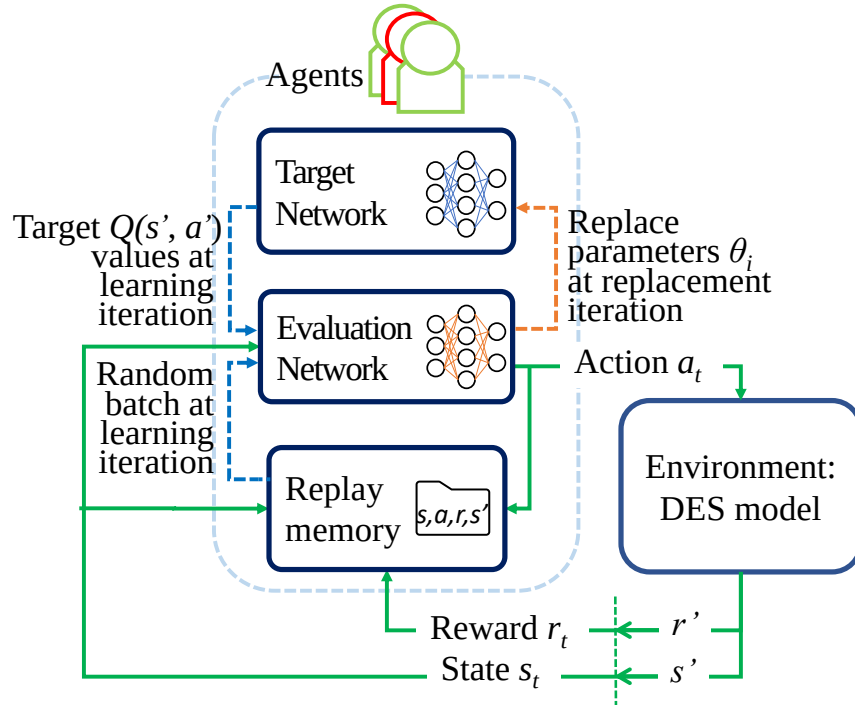


Figure 7.4 Agents learning process applying DQN for RL

7.5.5 Order release agents

Figure 7.5 shows the operation scheme of order release agents for each product. The agent acquires system's state s_t and selects when to start the next lot through its DQN evaluation network. The reward r_{t-1} of the past action is also passed to the agent by the model. Upon selection, agent holds according to the selected time, then it starts a new lot. The agent repeats the same procedure for the next lot start. There is a separate agent for each product type. Agents learn independently as they have separate DQNs.

State and reward formulation determines the quality of learnt policies. Policies can not be improved when state definition does not include all required supporting information to make decisions. Similarly, reward function formulation has an essential role as the correcting feedback from the environment or the model. Failing to present a comprehensive reward affects the learning capabilities of agents. In the following, we present state and reward formulation for order release agents.

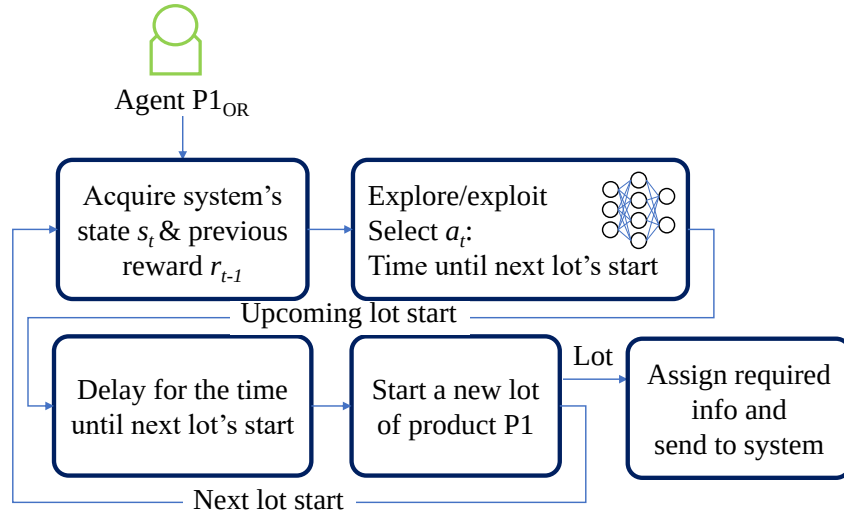


Figure 7.5 Working logic of order release agent for product type 1

State information for order release agents

State information for order release agents includes performance measures for system's resource, stations, and WIP. This state formulation permits the agent to perceive the system properly as related to its objectives. It permits agents to avoid overwhelming the system while respecting the due dates of job orders. The state includes information about other products' WIP. Hence, each agent considers the existence of other order release agents. They learn to avoid release times that may affect the global system's performance. Our state information for order release agents consists of the following:

- Average number of waiting lots in each queue
- Average total queue waiting times in each queue
- Average Cycle Time X factor (CTX) of all products' types. CTX: total time in system/ total value-added time
- SR of all products' types
- Completion% of process flow steps and ratio for the WIP of each product's type
- All equipment state: up/failed

All average values that are included in our state are moving averages. This gives agents proper understanding of the system through perceiving trends instead of whole averages or

instantaneous absolute values. As shown in the state definition, information that is related to the whole system and other products is provided for order release agents. This facilitates reaching a cooperative environment among managing agents. SR and completion% of process flow let the agents to perceive its position with respect to jobs' due dates. Through these information, the agent has insights about how fast or slow the order release should be. The agent may decide to slow down the order release if, for example, majority of its product's WIP is at the end of their process flow. Hence, yielding more system's availability for other order release agents whose products may be relatively behind schedule.

Reward function for order release agents

The reward for order release agents provides model's feedback for global system performance and for the specific product that each agent is responsible for. Order release agents are required to decide lot starting times considering the general performance and respecting due dates. Hence, the reward function is selected carefully to aid agents to learn adequate strategies considering its objectives. The reward function of order release agents, shown in Eqn. 7.2, includes the following:

- General average CTX for recently produced lots
- Average utilization of all system's equipment
- Throughput TH of agent's product type
- Job order completion/due date exceeding

$$r_{OR,t} = \frac{CTX_{base}}{CTX_{MA}} + Util\%_{overall,MA} - \frac{TH_{base}}{TH_{MA}} + JobOrder_{Slack} \quad (7.2)$$

Eqn. 7.2 shows the reward function for order release agents at time t . The reward consists of 4 components that differ in range. Hence, we adjust their relative weights through some base values that are denoted by the subscript *base*. As in state information, the average used in the reward are moving averages. Moving average terms have the subscript *MA*. The use of moving average permits agents to perceive trends. For example, using moving average for CTX for some recent finished lots is more suitable than using average of all lots.

The last component, $JobOrder_{Slack}$ provides the model's feedback about job order on-time completion or due time exceeding. The job order can be completed either before or after the due time. Completion after due time needs to be avoided. Also, too early completion should be avoided if it leads to due time exceeding of other job orders. Thus, we define the $JobOrder_{Slack}$ reward component for two situations, completion before and after due time. In case the completion is before due time, the agent gets positive reward if not earlier than an acceptable fraction of the available time for the job order, otherwise it gets negative reward. In our reward function, this fraction is set as 0.4. The available time for a job is calculated starting at when the previous job order was completed until the due time of the current job order. For early job order completion, an instant value is assigned for the $JobOrder_{Slack}$ component as follows:

$$JobOrder_{Slack,Early} = (0.4 - (DT_{job} - CT_{job})/AT_{job}),$$

where DT_{job} is the due time for a job order, CT_{job} is the actual completion time of a job order, and AT_{job} is the available time for a job order. If the due time is exceeded, the agent receives a negative reward, with each order release, according to the remaining quantity to fulfil the job order. In that case, a value is assigned for the $JobOrder_{Slack}$ component as follows:

$$JobOrder_{Slack,Overdue} = -Qnt_{Rem}/Qnt_{Total},$$

where Qnt_{Rem} is the remaining quantity to be produced to fulfil a job order, and Qnt_{Total} is the total quantity required. Upon job order completion after the due time, the agent gets an instant reward for completing the job according to how fast it could make this completion. The $JobOrder_{Slack}$ component is calculated for this case as follows:

$$JobOrder_{Slack,Delayed} = \frac{AT_{job}}{CT_{job} - DT_{job}} \cdot \frac{Qnt_{Rem, DueTime}}{Qnt_{Total}},$$

where $Qnt_{Rem, DueTime}$ is the remaining quantity to fulfil the job order at the due time.

7.5.6 Dispatching agents

Figure 7.6 shows the scheme of operation for dispatching agents. There is a separate agent for each equipment type station. As order release agents, dispatching agents are trained independently through separate DQNs. Dispatching agents start acting when a dispatching is possible/needed. This happens when the state of the station changes. Stations state change includes availability of an equipment that was failed, availability of qualified operator, and presence of waiting lots to be dispatched. Dispatching agents check resources, qualified operator and equipment, availability. Then waiting lots are checked for possible dispatching. Dispatching is not possible if there is a product or recipe restriction/dedication with respect to the available equipment. In addition, for timers starting steps, lots are not dispatched if any of the timers' processing steps' equipment is failed. If there is a dispatching possibility for any of the waiting lots, agents move to the next step for selection. If not, agents hold and retry upon its station state change.

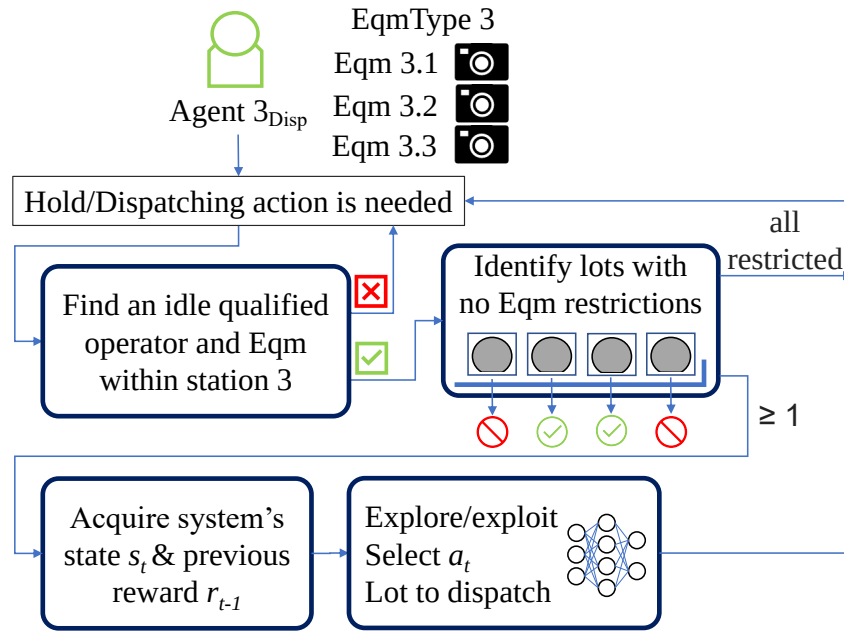


Figure 7.6 Working logic of dispatching agent for equipment type 3 station

Selection of a lot to be dispatched is done through agents' DQN. Agents either exploit their learnt policies, or explore and select a random action according to their ϵ value. Dispatching agents' actions include no dispatching. An agent may select not to dispatch for many reasons including filling the equipment capacity or yielding the equipment for another upcoming lot.

State information for dispatching agents

State definition is essential to improve stations' performance along with achieving collaborative policies. State is the channel through which the agent perceives its own station and other agents' stations as well. If the state does not include other stations' information, the agent focuses only on the its own station's performance. This may create bottlenecks and congestions or increase the waste. Our state definition for dispatching agents provides local and global details. The local details are for the waiting lots that are within agent's station as follows:

- Timer information: already within a timer, timer's start, or not
- Timer limit: time until timer's limit
- Lot type
- Product ID
- SR
- Number of wafers
- Lot's CTX

The global details that are included in dispatching agent's state are:

- Numbers of waiting lots in other station
- Each equipment state: up/down
- Number of lots arriving after (1-5) processing steps for each type of product

Through this state definition, dispatching agents learn to cooperate considering local and global objectives. Through the last three components of the state, each agent gets information related to the other agents' stations. Given a representative reward function that includes local and global objectives, collaborative behavior are reached. There are many situations in which agents must directly consider each other such as timers processing. The agent that starts a timer has to consider the level of congestion at next timer's steps stations and whether their agents are capable of handling that timer or not. Similarly, the other agents should also consider these upcoming lots that are within timers while dispatching other lots.

Reward function for dispatching agents

The model's reward for dispatching agents, as the state, is related to local and global objectives. Because of the collaborative behavior that is required from agents, a representative and fair reward function should be formulated. This is essential in case of rewards that are shared among more than one agent such as those related to timers limits exceeding. In such cases, it is not practical to assign a single value as reward. This is due to the different adequacy of each agent's action through timer's steps. When reward is selected to be a single value, it results in trivial actions such as not starting a timer to avoid timer limit exceeding. This implies the importance of reward formulation and assignment for reaching collaborative policies.

Our reward function for dispatching agents, shown by Eqn. 7.3, contains the following:

- The general average CTX for recently produced lots from any product type
- The average waiting time of lots in the agent's queue
- The average number of lots in the agent's queue
- The utilization level of the agent station's equipment
- The timers processing rating; representing the capability of this agent to process lots within timers without exceeding time limits
- The negative reward for exceeding timers which is assigned upon failing to respect a timer's limit

$$\begin{aligned}
 r_{Disp,t} = & \frac{CTX_{base}}{CTX_{MA}} + \frac{TQ_{wait,base}}{TQ_{wait,MA}} + \frac{NQ_{wait,base}}{NQ_{wait,MA}} \\
 & + Util\%_{MA} + \frac{TimRating\%_{base}}{TimRating\%} - TimReward_{exceed}
 \end{aligned} \tag{7.3}$$

Eqn. 7.3 shows the reward function for dispatching agents at time t . The first term, CTX_{MA} , is the same as in order release agents reward function. The second two components, $TQ_{wait,MA}$ and $NQ_{wait,MA}$, define the waiting time and number in agent's queue respectively. The fourth component, $Util\%_{MA}$, represents agent's equipment level of utilization. The last two components are related to timers' processing.

$TimRating\%$ is a rating for timer processing for that agent. It defines agent's capability of processing lots within timers. $TimRating\%$ is calculated as the ratio of number of wafers with exceeded timers to the total number of wafers within timers that were processed by that agent. An agent with a $TimRating\%$ value of 2% means that it 2% of wafers within timers, that were processed by that agent, had their timers' limit exceeded. Agents are seeking for reducing this value through their training phase.

$TimReward_{exceed}$ is instant reward with different values. It is assigned to agents whose steps involve an exceeded timer's limit. The value is related to agent's order among the timers' steps. For a timer whose steps involves agents A, B, C, and D, for example, there are 3 cases for a timer exceeding. Agent A, which has started the timer, gets a negative reward that is proportional to how risky it was to start that timer. The risk is calculated at the timer starting step according to the level of congestion of the next stations in which the lot has to be processed through its timers steps. The more congested the stations are, the more risky is to start that timer. Agents, such as B and C, which have already processed that lot get a negative reward proportional to the ratio of the lot's waiting time in their queues to their $TQ_{wait,MA}$. Agent which the timer was exceeded in its queue gets a negative reward similar to the previous agents and proportional to the ratio of remaining timer duration upon arriving in its queue to the timer duration. If the timer was exceeded in the queue of the timer's last step, the last agent's reward is doubled. However, no negative reward is assigned to any agent, if a timer was exceeded due to a failure of an equipment. If the lot was a rework for a previously exceed timer, all the negative rewards are doubled for all agents. This is because the timer's rework has only one chance of rework, otherwise it will be junked.

The $TimReward_{exceed}$ is calculated this way so that each agent gets a feedback that corresponds to its share in such consequence. Not all agents cause the timer to be exceeded, but some may have more responsibility than the others.

7.5.7 Interruption agents

Our conflicting agents are responsible for challenging the managing agents through interrupting systems resources. There are two conflicting agents: operator interruption agent and equipment interruption agent. When a working shift ends or equipment failure occurs, both operators and equipment become inactive. In our model, interruption and equipment failure or operators' end of shift are two distinct events.

Figure 7.7 shows the working scheme of operator interruption agent. This agent simulates what happen in real systems as operators get interrupted due to many reasons. Our interruption agent differs from reality as, through its DQN, it learns to interrupt the operator that

significantly affects the system's performance. This adds challenges to the managing agents, forcing them to explore new strategies to overcome the conflicting interruption agents.

The operator interruption agent starts by acquiring the system state and reward for its previous action. It selects an active operator to interrupt. If the operator is idle, the agent interrupts it and temporarily suppresses its availability. If the operator is not idle, loading or unloading an equipment, the interruption takes place after the operator finishes his/her task. This not only affects the availability of the operator for loading equipment, but also delays equipment unloading that were loaded by that operator. The interruption event, and its duration, are assigned randomly as shown in Table 7.1. The distributions are assumed based on the experience of the industrial team with experience in the real-world use case. After the interruption duration is finished, the agent holds for a random period of time before performing its next interruption. This random period is selected to follow an exponential distribution with a mean of 1 hour.

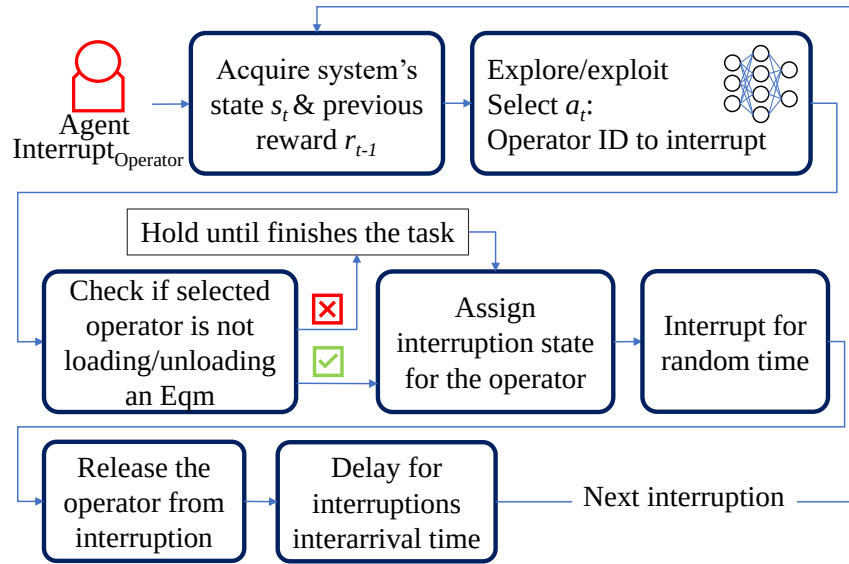


Figure 7.7 Operator interruption agent working logic

Table 7.1 Operator interruption events

Interruption event	Probability	Duration(min)
Discussion with an operator	10%	TRIA(0.5,4,6)
Discussion with an engineer	40%	TRIA(1,4,6)
Unplanned meeting	25%	TRIA(6,15,20)
Other	25%	TRIA(3,5,8)

Figure 7.8 shows the working scheme of the equipment interruption agent. This agent injects interruptions for processing equipment. This results in prolonging their unloading time, hence delaying the unloading operator and lot(s) that is(are) being processed. The agent starts by acquiring the state and last action's reward. Then, it selects an equipment among busy equipment for interruption. This is done through exploiting the learnt policy through agent's DQN, or through exploration. Upon selection, the unloading time for the equipment is increased by 20% of the lots' elapsed time within the equipment. This gives a chance for this agent to work with the operator interruption agent against the system. The effect of equipment interruption may increase if the operator who is responsible for unloading was unavailable. After the equipment is released, the agent holds for 10 minutes, simulation time, before its next interruption.

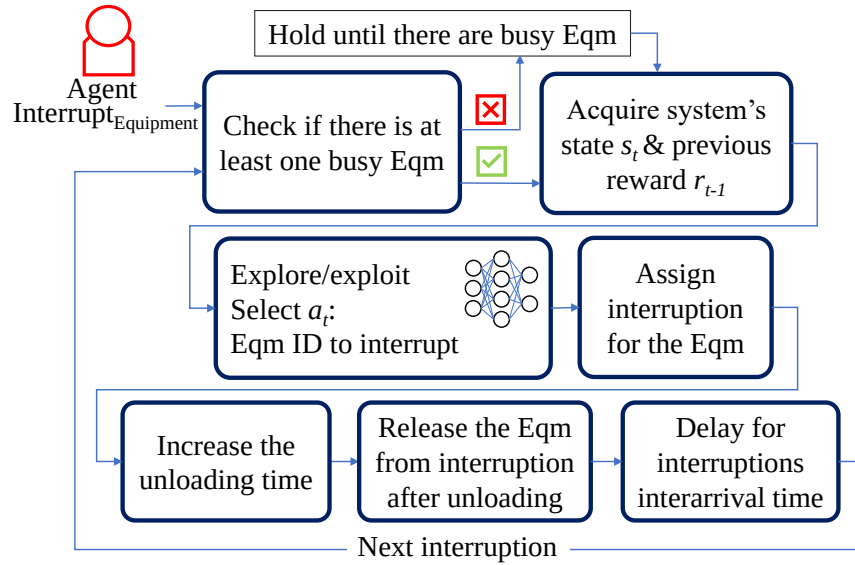


Figure 7.8 Equipment interruption agent working logic

State information for interruption agents

State information for operator interruption agent includes general information about the system and operators activity. The agent learns to select the operator that affects system's performance significantly according to the information of the state. State information for operator interruption agent consists of the following:

- Average number of waiting lots in each queue
- Average total queue waiting times in each queue

- Average CTX of all products' types. CTX: total time in system/ total value-added time
- Completion% of process flow steps for the WIP of each product's type
- All operators state: active/inactive

The state for equipment interruption is the same as the operator interruption agent's one except the last part. For equipment interruption agent, we pass equipment state instead of operators state. Through this state, agents select the operator/equipment to interrupt which affects local and global performances. Hence, dispatching agents are challenged by the operator interruption agent as their stations maybe delayed by the unavailability of qualified operators. Also, dispatched lots are subject for delays for the same reason when the equipment finish their processes. Similarly, the order release agents are challenged by the delays that operators and equipment experience through interruptions.

Reward function for interruption agents

The reward for interruption agents reflects how their interruption actions affected the performance of the system. The reward consists of 3 components related to products and resources. The reward function for interruption agents, shown by Eqn. 7.4, contains the following:

- General average CTX for recently produced lots of all products types
- Average utilization of all equipment
- Average throughput TH of all products types

$$r_{Interrupt,t} = \frac{CTX_{MA}}{CTX_{base}} + \frac{Util\%_{base}}{Util\%_{overall,MA}} + \frac{TH_{base}}{TH_{MA}} \quad (7.4)$$

As shown in Eqn. 7.4, agents are rewarded if they increase the CTX, or decrease the equipment utilization and production throughput. This is opposite to these components in managing agents' reward functions. Through simulation, both managing and conflicting agents have opposing objectives. This enrich their policies and continuously force them to explore new strategies. The effect of introducing conflicting agent is presented in Section 7.6.

7.6 Results and discussion

7.6.1 Use case overview

Our work is a continuation of a research project that is done for a real semiconductor manufacturing system. Semiconductor industry have many sources of complexities as discussed previously in Section 7.4. In our previous work in [1], we applied DQN for dispatching agents only. In this paper, we extend this application to include order release agents for each product type. Also, conflicting agents, for interruption of equipment and operator, are added to account for human operator presence, and to improve the learnt policies of the dispatching agents without prolonging training time. Our system has the characteristics of a complex job shop including high-mix of nonstandard products and the need for flexibility to handle this customer-centric scheme of production. It exhibits characteristics of both discrete manufacturing and process manufacturing. It relies on recipes, similar to process manufacturing, while also facing the complexities associated with discrete manufacturing. These complexities include long cycle times, make-to-order or engineer-to-order processes, intersecting process sequences, equipment reconfiguration for different recipes, and operators with various qualifications. The engineering-to-order production normally makes the system highly dynamic and induces many sources of variabilities. This includes the dynamics and variabilities that are created due to the essential presence of human operators in such system. The main objective of this research project is to build a decision-support tool. For a realistic implementation, that decision-support system must consider the presence of human operators given all their associated variabilities.

Presently in the real system, a hybrid rule is used for dispatching that combines a number of heuristics. The heuristics have a certain order and priorities for waiting lots are assigned accordingly. These heuristics and their order are as follows:

1. Earliest timer's limit
2. Relative importance given to products
3. Slack ratio
4. Creation time

The priority is assigned according to each rule respecting their order. This means that the first rule is applied first. If two lots have same priority according to a certain rule, the next rule in order is applied. A dispatched lot may result in dispatching less priority lots as

well if they have same recipe and the equipment have enough capacity. Timers are handled according to their level of risk. The risk of a timer is based on the ratio of total time of its processing steps to the timer's limit. If this value passes a certain threshold, the equipment of timer's steps are held in advance. Normally, no timer is started if one of its steps' equipment is failed. If a timer's limit is exceed, a timer's rework maybe applied. In that case, regardless the risk level, all equipment are held in advance for the timer's rework. In our RL setting, no hold-in-advance is permitted in any situation. In our model, Product 1 has 3 steps with a timer in two different fabrication areas. This timer's limit is 10 hours, while total processing time is up to 1 hour excluding transferring between fabrication areas.

Job orders, for each product type, have certain quantities and due times. Due to the variabilities within the systems, orders are usually overdue even though considering the SR among the dispatching rules.

System resources includes equipment and operators. Operators are responsible for loading/unloading the equipment that they are qualified to operate. Equipment are normally unloaded by the same operator who loaded them if he/she is still active. If that operator is loading another equipment, other equipment that need to be unloaded by him/her are delayed. Due to the engineering-to-order production, operators are subject to many interruptions that usually take place for activities related to production. This includes important discussions with the supervising engineering team for example.

Our implementation through simulation involves two different products with intersecting process flows passing by same equipment in different processing steps. There are a total of 25 stations in the model which products pass by through their process flow. Same equipment are utilized more than once within the process flow. Each product has an order release agent. Each station has an individual dispatching agent. For conflictions, we have two agents for operator and equipment interruptions respectively. In normal simulation with no RL, agents apply dispatching heuristics and random time intervals for order release, and conflicting agents apply random selection. Equipment up/downtimes are simulated through time distributions.

The implementation aims to improve the learnt policies of the agents within the same training time while considering realistic interruptions. A comparison is made between the agents' performance with and without the presence of conflicting agents, which represent operator and equipment interruptions.

7.6.2 Agents DQN configuration

Table 7.2 shows the DQNs' parameters for our managing and conflicting agents. The size of each agent's input layer corresponds to its state information as mentioned in Section 7.5.5, 7.5.6, and 7.5.7. For dispatching agents, they observe the information of 20 waiting lots which are processable for the given equipment.

The output layer size corresponds to the number of different action from which agents select. For dispatching agents, in addition to selecting from the 20 observed waiting lots, there is an extra action for no dispatching. For order release agents, the selected action, having values [0-20], determines the time until the next lot starts. We set these times to follow the relation shown in Eqn. 7.5:

$$TTNR = 80 + 16 \cdot action(min), \quad (7.5)$$

where $TTNR$ is the time till next release. This relation, given action value [0-20], results in a minimum of 80 minutes and a maximum value of 400 minutes for the $TTNR$.

For conflicting agents, they select a busy equipment and an active operator for equipment and operator interruptions respectively. In our model, we have a total of 31 equipment and 40 operators with different working shifts. We use a masking layer for actions of dispatching and conflicting agents to avoid non-applicable actions such as selecting non-existing lot, or interrupting an inactive operator/idle equipment.

Table 7.3 shows the parameters for DQN training. Each agent has its own replay memory. At each learning iteration, after a certain number of actions, a random batch is drawn from agent's memory for training of the evaluation network. The evaluation network is copied to replace the agent's target network at replacement iteration. Each agent type has its specific learning parameters as they differ in the action taking frequency. Similarly, the DQN parameters are also specific for each agent.

7.6.3 Results

We execute our model to test the performance of our DQN agents in a simulated environment. The performance is compared mainly between normal system, and system where RL is implemented. In normal case, heuristics are applied and operator/equipment are selected randomly for interruptions. While for system where RL is implemented, DQN managing agents are responsible for order release and dispatching. Operator/equipment interruptions

Table 7.2 DQN's parameters for managing and conflicting agents

Parameter	Dispatching	Order release	Equipment interruption	Operator interruption
Input layer size	176	107	105	114
Output layer size	21	21	31	40
Number of hidden layers	2	2	2	2
Hidden layer 1	140	85	84	93
Hidden layer 2	88	53	52	57
Activation function	ReLU	ReLU	ReLU	ReLU
Exploration probability	0.15	0.15	0.15	0.15

Table 7.3 Learning parameters for managing and conflicting agents

Parameter	Dispatching	Order release	Equipment interruption	Operator interruption
Replay memory size	20000	1000	2000	2000
Minimum memory size	1000	200	200	200
Learning batch size	1000	150	200	200
Learning iteration	40	10	40	40
Replacement iteration	320	100	200	200
Learning rate	0.005	0.002	0.005	0.005
Discount factor	0.85	0.90	0.85	0.85
Optimizer	Adam	Adam	Adam	Adam

are also handled through DQN conflicting agents in the RL setting. In addition, we compare between system with RL when DQN conflicting agents are activated and when there is no interruptions as in [1]. This is done to prove that DQN agents' performance improves when they experience conflixtions. The model starts with an empty system and no failed equipment. All agents start with random policies. Through the simulation, they learn through the model to improve their policies.

Figure 7.9 shows the order release times in normal system applying interarrival times-based releases following specific time distributions. The same order release policy is applied for system with DQN dispatching agents that is presented in [1] with no interruptions. Product 1 and 2 are denoted as $P1$ and $P2$ respectively. Figure 7.10 shows the order release times in the system with DQN managing and conflicting agents. The figures show that the DQN order release agents generally started more lots, within the same simulation time, than the applied interarrival times-based order release. The order release agents learnt policies to start more lots with specific timing. This normally adds more pressure on the system. However, the cooperation between order release agents and dispatching agents helped increasing the throughput without causing congestions or increasing the waste. This should be considered

when comparing different systems. The system with DQN managing and conflicting agents is more pressured by more order releases than the normal system and the system with DQN dispatching agents with no interruptions.

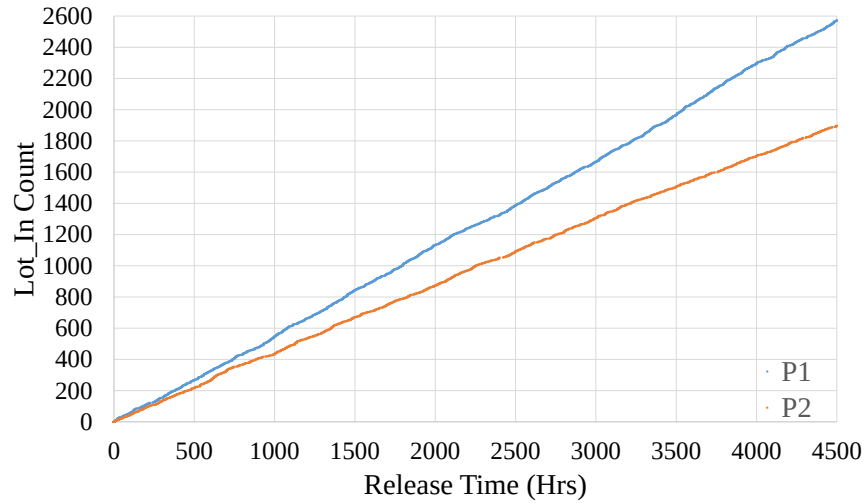


Figure 7.9 Order release times in normal system applying interarrival time distributions

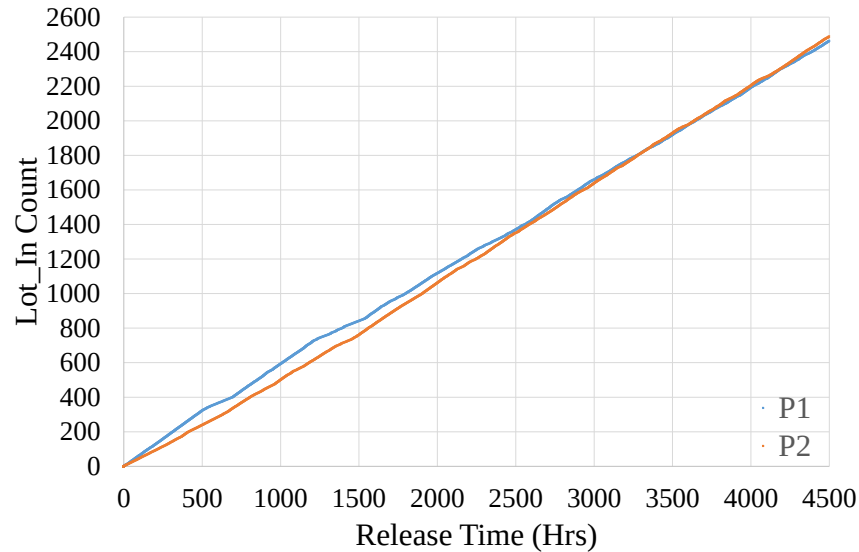


Figure 7.10 Order release times in system with DQN managing and conflicting agents

Figures 7.11 and 7.12 show the CTX in normal system applying heuristics, and system with DQN managing and conflicting agents respectively. The CTX is a measure that relates lots' total time in system to the value-added time. The less CTX the better. It means less non-value added time that lots have through the system. We smooth the CTX values using moving average for the last 50 produced lot for each product. In normal system, the CTX

shows an increasing trend which means that the system can not handle its WIP normally and congestions are created. However, system with DQN managing and conflicting agents keeps the CTX at lower and non-increasing levels.

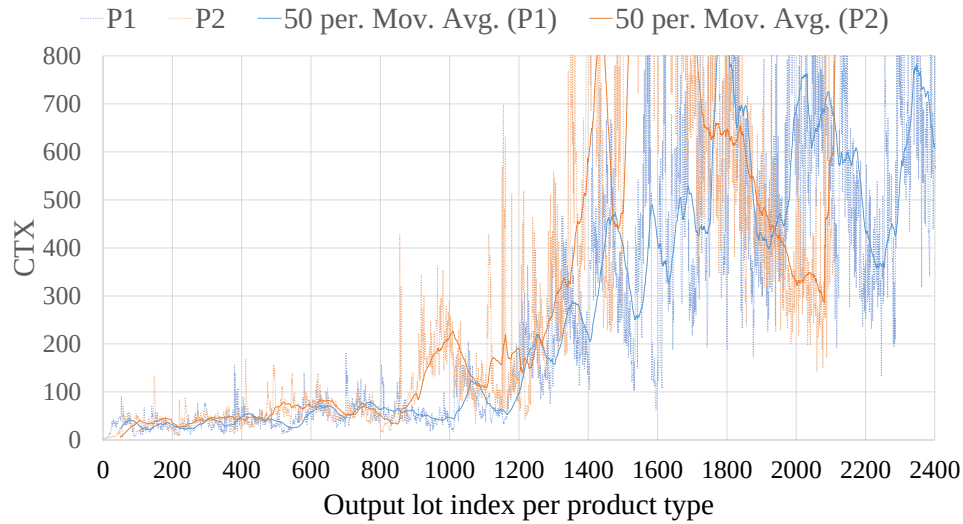


Figure 7.11 CTX in normal system applying heuristics and random operator/equipment interruptions

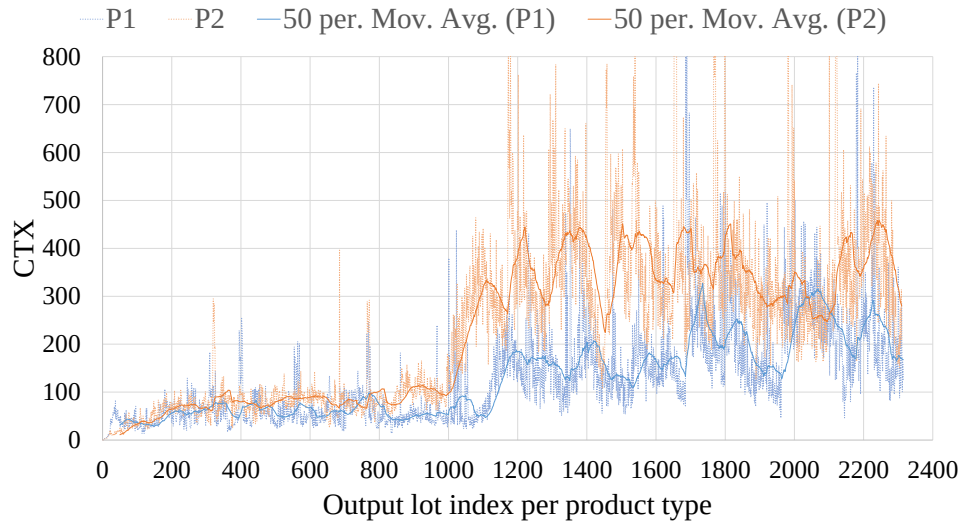


Figure 7.12 CTX in system with DQN managing and conflicting agents

Figures 7.13, 7.14, and 7.15 show percentages of rework and junk lots that are processed within a timer with respect to total lots that entered the system. Figure 7.13 shows that normal system's level of exceeded timers' limits is high and increasing as compared to systems with DQN shown in Figures 7.14 and 7.15. Moreover, the presence of conflicting agents shows

an improvement in the learnt policies. The rework and junk levels are almost halved when the agents were conflicted by the presence of interruption agents. This happens even though that system is more pressured by lot starts that are made by order release agents as compared to the interarrival times-based order release in the other cases.

Figures 7.16, 7.17, and 7.18 show each product's throughput in normal system, system with DQN dispatching agents only and no interruptions, and system with DQN managing and conflicting agents. Generally, the throughput of the normal system is the lowest due to the high level of congestions that it experiences over time. The throughput of the system with DQN managing and conflicting agents is better than that of the system with DQN dispatching agents only and no interruptions.

Figures 7.19 and 7.20 show the job orders' relative earliness/tardiness in normal system and system with DQN managing and conflicting agents respectively. The relative earliness is calculated at quantity production completion of each job order. It is calculated as the ratio of the time till/since the job orders due time to the total time that was available for that job order. A negative value indicates that the job order was fulfilled after the due time. As shown in the figures, the normal system can not fulfill job orders in time when congested. In contrast, the system with DQN agents made almost all job orders before their due dates. For product 2, jobs were fulfilled relatively earlier as agents decided that there is possibility of increasing the throughput of that product without affecting the rest of the system.

Through the state definition and the comprehensive reward functions, the managing agents could reach a cooperative behavior. The dispatching agents could cooperate with each other. In addition, the order release agents achieved a proper balanced lot start times to improve the system's throughput and on-time job orders fulfilling without overwhelming the dispatching agents with short times between lot starts.

The presence of conflicting agents for interruptions added a challenge for the managing agents. Interruption agents also learnt to cooperate to actively affect the system's performance. This showed the managing agents situations that they would not have seen frequently in the simulation, hence forcing them to explore better policies. This is clear for dispatching agents in handling timers.

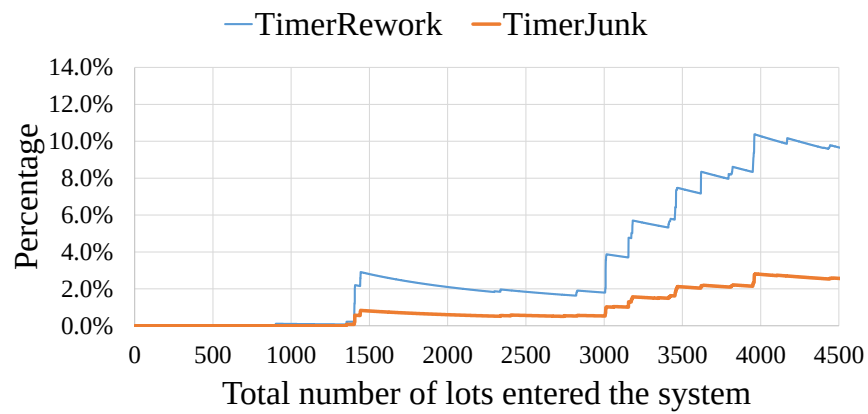


Figure 7.13 Timers rework% and junk% in system applying heuristics and random operator/equipment interruptions

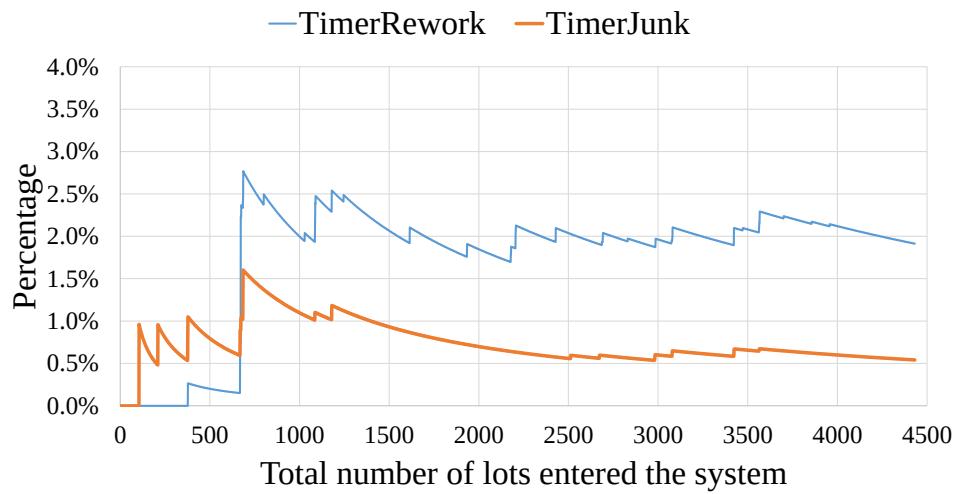


Figure 7.14 Timers rework% and junk% in system with DQN dispatching agents only and no interruptions

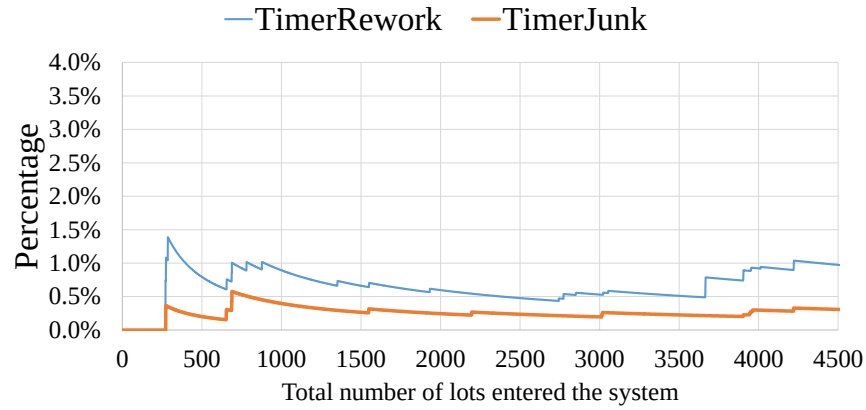


Figure 7.15 Timers rework% and junk% in system with DQN managing and conflicting agents

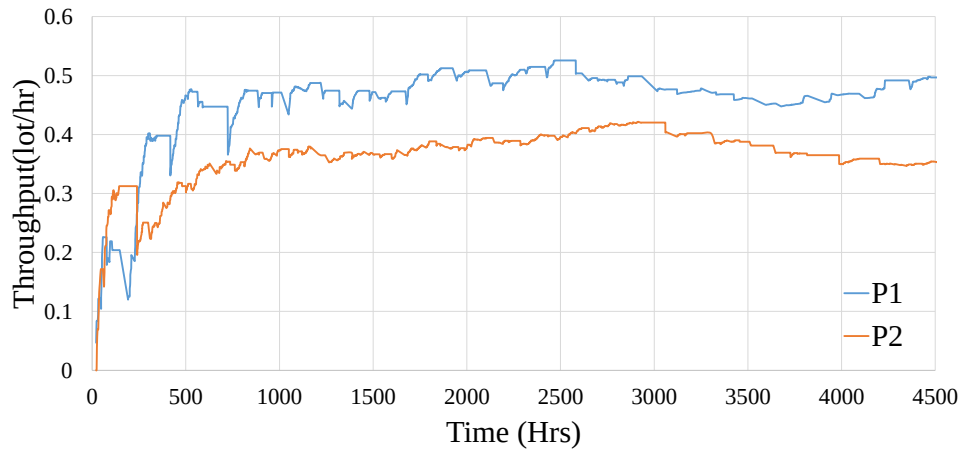


Figure 7.16 Throughput of system applying heuristics and random operator/equipment interruptions

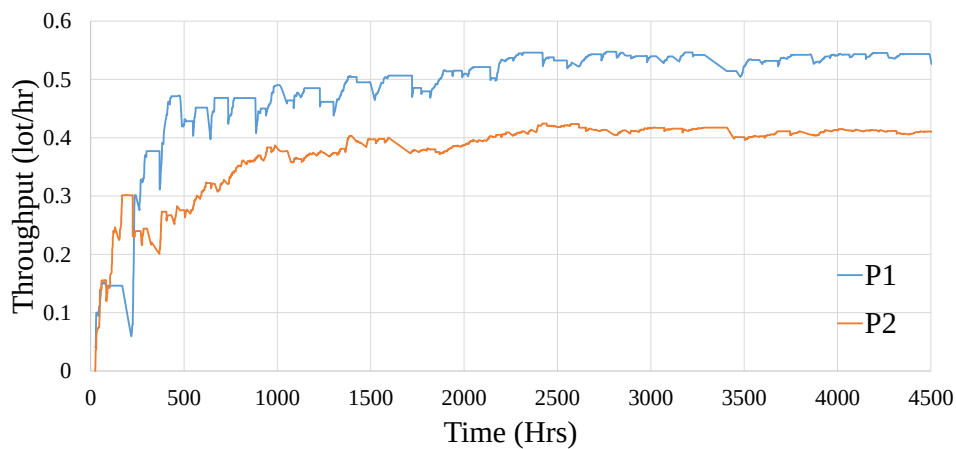


Figure 7.17 Throughput of system with DQN dispatching agents only and no interruptions

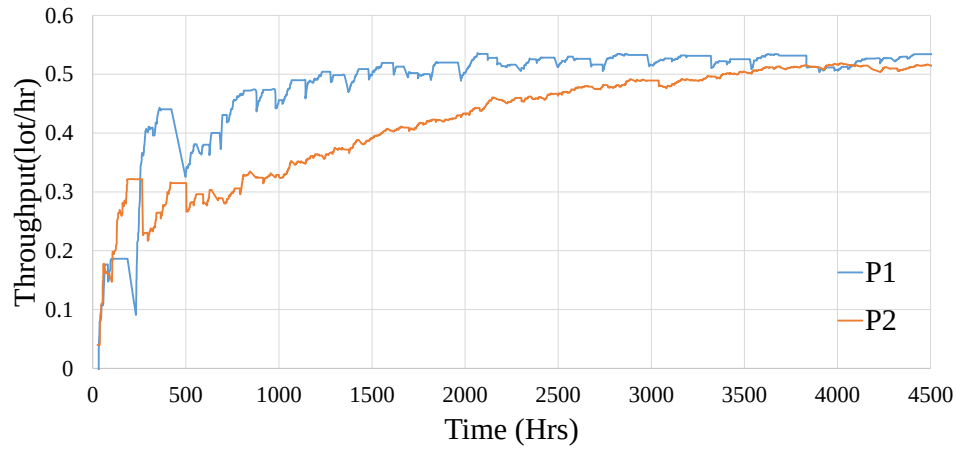


Figure 7.18 Throughput of system with DQN managing and conflicting agents



Figure 7.19 Job orders' relative earliness/tardiness in system applying heuristics and random operator/equipment interruptions

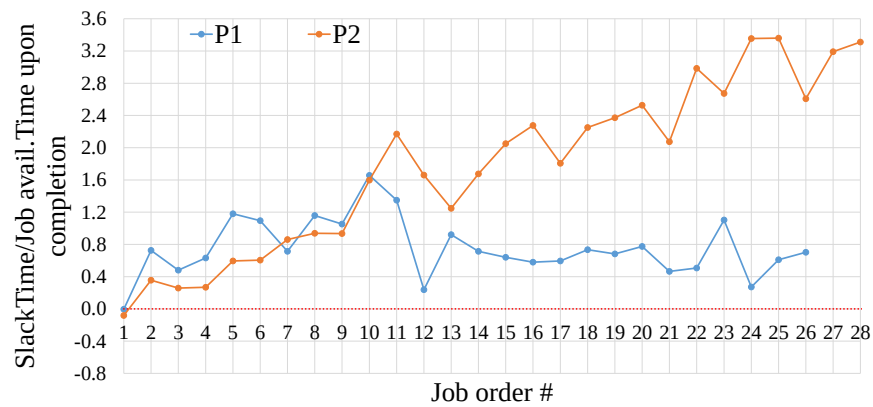


Figure 7.20 Job orders' relative earliness/tardiness in system with DQN managing and conflicting agents

7.7 Conclusion

In this paper, we enriched the learnt policies of managing agents which are responsible for dispatching and order release. This was achieved through the introduction of RL conflicting agents following the concept of self-play. The managing agents policies were improved without extending the training as compared to training with no interruptions. The conflicting agents were implemented for operators and equipment interruptions replacing random interruptions that are traditionally implemented in simulation. This ensured high impact of the interrupted operator/equipment on the system. Thus, the conflicting agents actively forced the managing agents to explore different policies to overcome the interruptions effect. For a production system, the introduced interruptions account for similar events that take place in reality. This enhances the readiness of managing agents policies to be implemented in real systems considering the presence of human operators.

We compared our strategy to system that included managing and conflicting agents to normal system applying heuristics and system with RL dispatching agents only with no interruptions. The later comparison detects the difference in learnt policies that is achieved by the conflicting agents presence. The managing agents learnt to cooperate to improve the general system's performance. Order release agents learnt to achieve proper lot release times avoiding overwhelming the dispatching agents. In addition, conflicting agents learnt to cooperate together against the managing agents interrupting the operator and the equipment that have high impact on the system.

Our results show an improvement in managing agents' policies when conflicting agents were introduced. The improvement was achieved within the same training time for managing agents with no interruptions. This enhances the applicability of such RL-based decision-making in real system considering variabilities caused by human operators and considering their indispensable presence.

Acknowledgements

This work is funded through the Mitacs Accelerate program, Grant No. IT01632, in partnership with Teledyne DALSA Semiconductor, a business unit of Teledyne Digital Imaging, Inc. The authors would like to thank the industrial and information technology teams at Teledyne DALSA semiconductor for their great support and collaboration.

**CHAPTER 8 ARTICLE 5: ADAPTIVE DISPATCHING AND ORDER
RELEASE THROUGH SELF-PLAY DEEP REINFORCEMENT LEARNING
CONSIDERING DYNAMIC DEMAND AND PRODUCTION CONDITIONS**

Ahmed H. Sakr, Ayman Aboelhassan, Soumaya Yacout, Samuel Bassetto

Submitted to:

Computers & Industrial Engineering

December 23, 2022

8.1 Abstract

For production systems, agents are considered players that make decisions such as dispatching activities. They aim at winning against their opponents which are customer orders and processing challenges such as resources interruption. Their winning is achieved through fulfilling customer orders without tardiness. According to the difficulty of these challenges, agents learn to improve performance based on their Key Performance Indicators (KPIs)-related reward. Such performance may decrease upon experiencing real situations such as variable demand.

In this paper, we introduce Reinforcement Learning (RL)-based customer agents which vary their demand rate with each job order they place. This is applied in a game-like scheme in which customers objective is to challenge the system by reaching orders completion within a certain delay range. Through this self-play scheme, we create challenging customer agents, representing the opponents, that learn to challenge the managing agents. Along with customer agents as main opponents, we add other conflicting RL agents that create resources interruptions. Thus all agents, including the opponents, keep improving their policies through training. This strategy considers the confrontations through different production conditions that exist in reality. Through training, such situations are exaggerated to expose agents to difficult cases within the limited training time. Our proposed approach is applied on a use case for a real semiconductor manufacturing system.

The results show promising performance of the managing agents in such high variability situations. Job orders are mostly completed before their due dates that are set by the customer agents. The order release agents adjust their release times so that jobs are not tardy and the system is not overwhelmed with excessive Work-In-Process (WIP). This shows the readiness of the managing agents to tackle such situations in the real system without the need for prolonging training times.

8.2 Introduction

Production management systems encounter increasing challenges due to uncertainties within production systems. The movement toward customer-centric markets and the need for fast adaptability are among the reasons for such variabilities [115]. The Industry 4.0 (I4.0) initiative was launched to provide the required flexibility for production systems [116]. Through the I4.0 and its associated Internet of Things (IoT) technologies, data collection and communication are achieved. This is essential for decision-making strategies giving competitive opportunities for these production systems. Accordingly, these systems thrive agility and

suitable response to different situational changes. For instance, Machine Learning (ML) identifies hidden and complex patterns not considered by traditional strategies which leads to performance improvement [117].

Reinforcement Learning (RL) is one field of ML algorithms that is used for decision-making based on learning policies in a semi-supervised scheme. It learns through interaction with its environment, applying decisions, and receiving feedback for decisions' adequacy [21]. For providing the required training environment, Discrete-Event Simulation (DES) has been utilized for its capabilities to represent both stochastic and dynamic behaviors [1]. In this case, the RL algorithms are trained in an off-line mode before being deployed in the real system. This is done as such algorithms need hundreds of thousands of training episodes, making it impractical to be trained in an online mode.

Due to relying on simulated environment, the performance of the RL learnt policies is limited to the model's abstraction and the frequency of different challenging situations. Such situations maybe injected into the simulation randomly. However, their randomness does not consistently impact system's performance upon occurrence. The need for resilient decision-making in challenging situations was clear during the COVID-19 crisis. The crisis showed the importance of valuing flexibility, shortening response times, and avoiding focusing on costs only [118]. The COVID crisis has impacted the upstream and downstream supply chains globally [119]. Due to its lean and globalized structures, supply chains are prone to disruptions such as the pandemic [120]. In semiconductor industry as an example, most of manufacturers suffered from lead time increase, alternation in overall demand, and order fulfillment tardiness [121]. In a framework where RL is employed for decision-making, such situations should be considered during the algorithm's training. Thus, ensuring proper performance in these situations when decisions are needed.

In this paper, we develop a method to train RL decision-making agents for a dynamic demand in a self-play scheme. We employ the Deep-Q-Network (DQN) as the RL algorithm for the agents. The training environment is provided through a DES model. The contributions involve achieving self-play scheme, as in two-player computer games, while providing realistic scenarios for the production system and avoiding trivially selected actions. This is achieved through employing other RL agents representing customers, opponents, which work against the managing agents that are responsible for the system's dispatching and order release. Accordingly, the managing agents are exposed to dynamic order's due dates that are set by the customer agents. For customer agents, trivial actions, such as selecting only the tightest due dates, are avoided through their reward function definition. We refer to the customers as the challenging agents as they challenge the managing agents through their selected due dates.

Further, the managing agents are subject to interruptions through two other agents referred to as conflicting agents. These agents select and interrupt system's resources, operators and equipment, that are expected to have high impact on the system's performance. This self-play scheme enforces that the managing agents actively seek to explore new strategies. Same applies for managing and conflicting agents. This puts the managing agents in challenging situations, showing their readiness for such conditions in reality. We apply the approach through a use case for a real semiconductor manufacturing system. This self-play scheme, including the introduction of the customer agents, is utilized as an extension for our preceding works in [1].

The paper organization is as follows: Section 8.3 presents a background for RL implementation in production systems. Section 8.4 states the problem of relying on random assignment demand rate for creating variable demand. It also shows the implementation challenges of the self-play concept in production systems. Section 8.5 presents the methodology for employing RL for managing, conflicting, and challenging agents. Section 8.6 presents the results of the proposed approach through a DES of a semiconductor manufacturing system use case. It shows how managing agents handle the dynamic demand without creating excessive job orders tardiness. It also shows the improvement in performance of the dispatching agents as compared to the same agents in our previous work in [1] in which no system interruptions were present. Finally, Section 8.7 presents a conclusion for results.

8.3 Background

Neural networks-based RL achieved noticeable success for Google DeepMind's AlphaGo project [25]. Through the communication technologies provided through the IoT, data-driven decision-making is enabled for RL applications. This supports adaptive and responsive decision-making for systems seeking for productional flexibility [122]. This success is transferred for application in production systems.

In production management field, the motivation for applying RL-based decision-making is the inefficiency of traditionally applied methods in complex systems [123]. In addition, these methods rely on expert knowledge which is prone to human error [124]. In comparison, RL methods are adaptive and does not suffer from static modeling problems.

In production scheduling, RL-based decision-making was applied to overcome the uncertainties caused by production customization. Lin et al. applied DQN for selecting dispatching rules for job shops scheduling [67]. They applied their approach to a semiconductor manufacturing system in which each machine was assigned with a specific dispatching rule. Their

approach showed performance improvement as compared to applying single heuristic. Altenmuller et al. employed a DQN agent in a process-based DES environment for dispatching decisions with time constraints [68]. They applied their approach for a complex semiconductor use case. Their results showed promising results as compared to competitive heuristics. Marc-Andre and Fohlmeister proposed an approach for avoiding local optimization tendencies for multiple RL agent system [125]. They considered both global and local performance objectives. They improved global logistics performance while benefiting from the distributed agents for real-time response. Mowbray et al. presented a RL-based methodology to address a single stage, parallel batch scheduling problem [126]. They considered constraints imposed through the operating procedures such as sequencing constraints. Their results showed that the proposed RL approach handled the constraints and uncertainties effectively as compared to mixed-integer linear programming approaches. Zhang et al. applied RL through multi-agent manufacturing system in which each agent is responsible for a single equipment scheduling tasks [127]. They considered stochastic job insertions and equipment failures to test agents' readiness for such situations. Their approach showed promising results in terms of dynamic responsiveness for disturbances as compared to traditionally applied heuristics. Shi et al. introduced RL for scheduling of a discrete automated production line including a DES for providing the training environment [128]. Their approach employed an agent for choosing transferring actions. Their results showed competitive performance and robustness as compared to heuristics in terms of processing time.

The utilization of RL for decision-making tasks proved its capacity to outperform conventional algorithms in most cases. This was shown through its ability to adapt to various scenarios and handle uncertainties. However, the real-world implementation is still hindered due to some challenges [116]. These challenges include handling unseen scenarios that appear rarely or do not appear through training. For real systems, complexity causes the state space to grow. This increases the computational efforts required for training in case of employing a single agent. Hence, decentralized decision-making, involving multiple agents, is recommended for tackling this problem [129]. However, local and global objectives must be considered for these agents to avoid local optimization for each agent.

Through the literature, one of research gaps is the generality of agents' policies. In order to develop RL-based decision-making that is applicable in real systems, these algorithms must be trained to handle different changes of production conditions. Such ability requires agents to adapt effectively to such variability. Methods for fulfilling this need are necessary considering limiting the training time to avoid other problems such as performance plateaus. In this study, we propose a self-play scheme applied for multi-agent in a complex production

system. The agents are responsible for managing dispatching and order release tasks. The implementation adopts the concept of decision-making decentralization in which each agent is trained individually with its own DQN and replay memory. The self-play scheme establishes opposing objectives between managing agents and both conflicting and challenging agents so that they actively seek different policies to outperform each other. This scheme generates changing conditions that pushes the trained policies to be improved.

8.4 Problem description

8.4.1 Problems of randomly assigned demand

Applying RL for decision-making support involves selecting certain actions. The selected actions are expected, by the algorithm, to be the proper ones according to the defined reward function. Action selection relies on the acquired system state information upon making decisions. The managing agents seek for improving system's Key Performance Indicators (KPIs) through selecting when to start lots and which waiting lot(s) to be dispatched at each equipment station. In semiconductor industry, a lot refers to a cassette including 25 wafers.

Production systems are stochastic and dynamic in nature. Demand variation elevates the dynamics of the system as the production rates are required to be continuously adaptive. In a RL context, this challenges the decision-making policies through moving targets and dynamic states. These challenges are due to the varying Slack Ratio (SR). The SR relates the remaining time till the due date to the remaining quantity to be produced for fulfilling the job order requirements. Thus, when the demand changes, the SR of the Work-In-Process (WIP) changes accordingly. The managing agents may need to accelerate a certain product type due to its low SR. Upon job completion or another product type's demand change, a product type of a certain priority may become relatively less important.

Dynamic demand can be simulated through randomly assigning due dates for job orders. Following this strategy, the managing agents may encounter some challenging situations. However, such randomly assigned due dates does not consistently challenge the system. The randomization does not take into consideration the state of the system and what the managing agents have learnt. It does not actively challenge the managing agents as they can be selected such that the SR is mostly high. Hence, within the limited training time, the agents may only learn a certain policy to improve the KPIs generally without addressing the problem of the varying demand. Extending training times can be thought of as a solution, yet it is not a practical solution as it does not guaranty better learning.

8.4.2 Self-play implementation challenges

In computer games, the self-play scheme has proven success for RL algorithms [130]. This success was shown in games where two opponents try to defeat each other. Both computer games and industrial applications share similar combinatorial problems that represent real-world optimization challenges such as manufacturing control [131].

The application of a self-play scheme for agents representing customers and production managing agents is not a direct task. In the real world, customers normally do not seek for defeating their manufacturing partner. They may set relatively tight due dates, but they do not aim at reaching tardy deliveries. To achieve realistic scenarios through the customer agents, their self-play working scheme has to consider challenging the system without resulting in excessively tardy job orders. Such realistic scenarios are essential for encouraging the managing agents to learn and improve their policies rather than confronting them with excessively hard situations. For agents learning through RL, such hard situations may be inevitable if they persist regardless of the system's state. If the objective of customer agents is to result in tardy deliveries with no bounds, their policies will trivially select the tightest possible due date for all job orders. In a RL context, this requires specific formulation of the reward function considering certain bounds. Such function rewards the customer for causing a definite range of tardiness through its selected due dates.

8.5 Methodology

In the proposed approach, we develop a DES model, following agent-based modeling technique, to provide the required training environment for the RL agents. Hence, the agents develop their decision-making policies through interacting with each other and with the system through the simulation. Agent-based modeling is suitable for training RL agents as it offers the required capabilities for the agents such as gathering system's state information, applying selected actions, and acquiring feedback for the past actions. More details for the utilized DES model are found in our previous study [1].

8.5.1 DQN agents within DES

Through DES, RL agents learn their decision-making policies through interacting with the simulated environment. The employed RL algorithm for agents is DQN. Figure 8.1 shows the learning process for RL agents including all agent groups. Through the simulation, decisions are required from different agents. Upon requiring a decision at time t , the simulation passes the state s_t following the state formulation of the agent that corresponds to the

required decision. According to the agent's policy, it selects an action a_t and passes it to the simulation for application. The simulated environment continues running the simulation and repeats the same when a decision is needed. For the same agent, when a decision is required from it, the simulation passes a reward r_t along with the new state s' . This reward is the environment's feedback for the agent's previous action representing its adequacy. Through the DQN algorithm, state-action-reward-next state components are stored in a replay memory. These components are used by the algorithm to train and improve the decision-making policy of the agent. In the system, each agent has its own policy, or DQN, and replay memory. Hence, each agent gets trained independently. This follows the concept of decision-making decentralization.

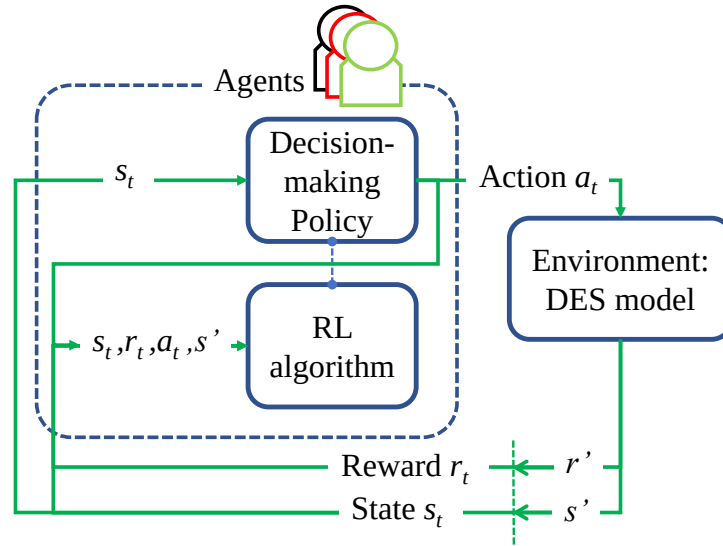


Figure 8.1 RL agents learning process within the DES model [1]

8.5.2 Overview for agents within the DES model

There are 3 groups of agents in the model which differ in functionality and objective; managing agents, conflicting agents, and challenging agents. We refer to agents that are responsible for dispatching and order release as managing agents. There are conflicting agents which cause interruptions to the resources, operators and equipment, to represent several interrupting situations that take place in real systems. The last group of agents are customers. Customer agents set the due date, or time, for which the next job order is required to be fulfilled. Figure 8.2 shows the functionality of each agents group within the DES model. The managing agents learn and employ their policies so that the KPIs are improved. The conflicting agents seek for the exact opposite. Upon their action selection time, they inter-

rupt the operator and the equipment which are expected to have high impact on system's performance.

Each customer agent set due dates for job orders of its product type. Within the proposed self-play scheme, customer agents are considered the main opponents for the managing agents. They set due date in which the system fulfills the orders within a certain range of tardiness. They learn to reach adequate policies so that they are capable of making such decisions according the system state. The following subsections present details for RL agents learning process. Also, the state information and reward functions for each agents group are presented.

8.5.3 Managing agents: dispatching and order release agents

The managing agents are order release and dispatching agents. There is an order release agent assigned for each product type. Similarly, there is a dispatching agent assigned for each equipment station. An equipment station refers to a group of equipment which have same type or similar capabilities. They may differ in capacity or speed.

Order release agents

For each product type, the order release agent selects the delay till the next release or lot start. The agents learn to select such delays so that the system's throughput is improved and the due dates for job orders are respected. This is done considering not to overwhelm equipment stations by an excessive WIP. Due to having a separate agent for each product type order release, an agent should consider the existence of other order release agents not to affect each other's performance. Thus, we include global information in the state and the reward rather than information related only to the agent's product type. The following list shows the component of an order release agent's state:

- Average number and time in queue for equipment stations
- Average Cycle Time X factor (CTX), total time in system/ total value-added time, of all products' types.
- SR of all products' types
- Completion% of process flow steps and ratio for the WIP of each product's type
- All equipment state: up/failed

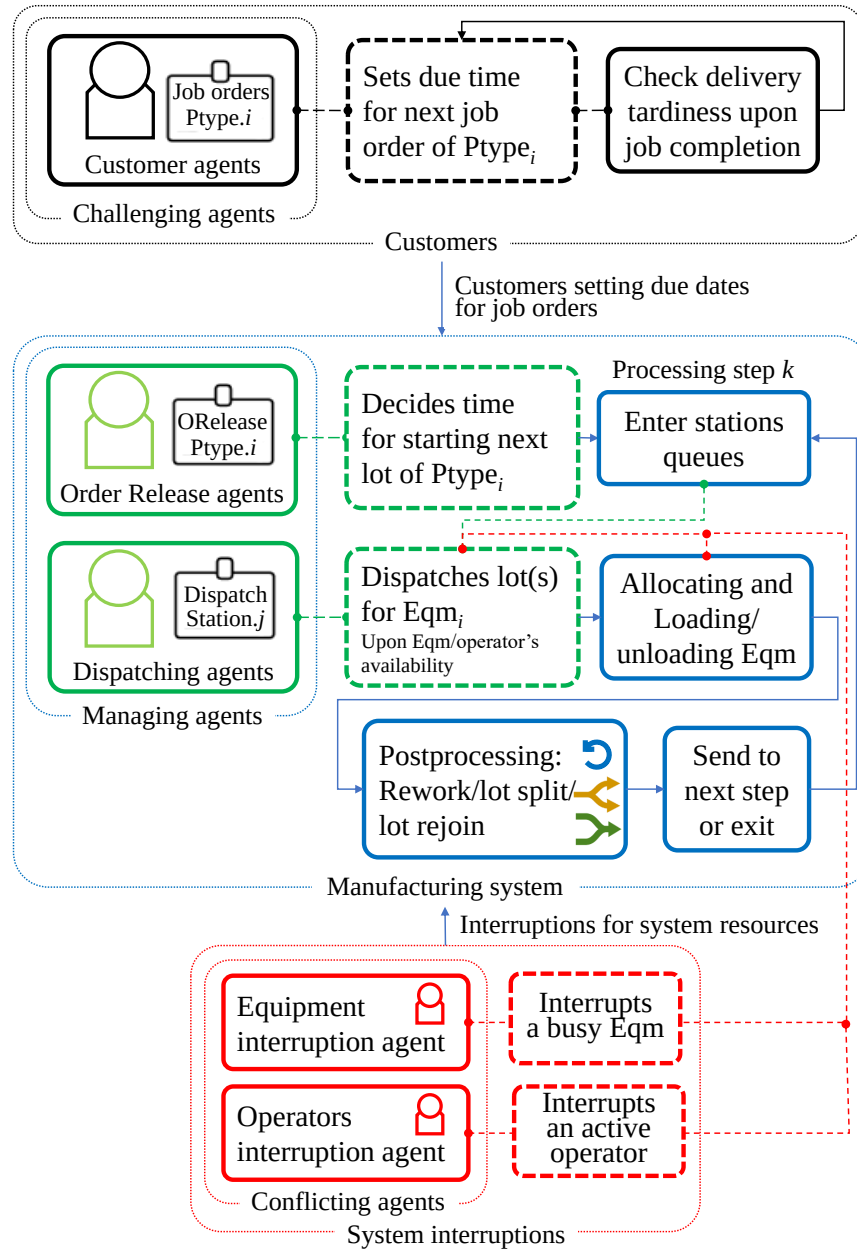


Figure 8.2 Functionalities of the agents within the DES model

Average values in state information are calculated as moving averages. This provides trends for the agents which are more insightful than whole averages or instant values.

The adequacy of order release agents' actions is obtained as the simulation's feedback through the reward function for these agents. The adequacy is related to the objectives required from these agents. The reward function at time t , shown in Eqn. 8.1, includes the following components:

- Average CTX for recently produced lots of all types
- Average utilization, $Util\%_{overall}$, of all equipment
- Throughput, TH , of agent's product type
- Job order advance/retard completion status

$$r_{OR,t} = \frac{CTX_{base}}{CTX_{MA}} + Util\%_{overall,MA} - \frac{TH_{base}}{TH_{MA}} + JobOrder_{Slack} \quad (8.1)$$

In Eqn. 8.1, we normalize the values of some terms due to the different ranges they have. This normalization is done through some base values denoted by the subscript *base*. These base values set their associated terms relative weights as well. The subscript *MA* denotes the use of moving average. $JobOrder_{Slack}$ represents the feedback for job order completion tardiness. As job order can be completed either before or after the due date, we define two formulas to calculate this component's value for each case. In case of early completion, an instant value is assigned for $JobOrder_{Slack}$ as follows:

$$JobOrder_{Slack,Early} = (\beta - (DT_{job} - CT_{job})/AT_{job}),$$

where DT_{job} is the due time for a job order, CT_{job} is the actual completion time of a job order, and AT_{job} is the available time for a job order. The reward value for an early job order completion decreases if it approaches a certain percentage β of AT_{job} . In this work, we set this value to be 0.4.

When due date is exceeded, the agent gets a negative value for $JobOrder_{Slack}$, denoted as $JobOrder_{Slack,Overdue}$, with each action until the job order is completed. This value is calculated as follows:

$$JobOrder_{Slack,Overdue} = -Qnt_{Rem}/Qnt_{Total},$$

where Qnt_{Rem} is the remaining quantity required to fulfil a job order, and Qnt_{Total} is the total quantity required. When such overdue job order is completed, the agent get an instant

value for $JobOrder_{Slack}$ that represents how fast it could rectify this situation considering the percentage of quantity of shortage at due date. This value is calculated as follows:

$$JobOrder_{Slack,Delayed} = \frac{AT_{job}}{CT_{job} - DT_{job}} \cdot \frac{Qnt_{Rem, DueTime}}{Qnt_{Total}},$$

where $Qnt_{Rem, DueTime}$ is the remaining quantity to fulfil the job order at the due time.

Dispatching agents

Dispatching agents are trained for selecting lot(s) for dispatching at each equipment station. They also have the option not to dispatch any lot, which maybe a suitable action in some situations. As for order release agents, each agent has its DQN that is trained independently through its own replay memory. The objectives of dispatching agents include improving their station's utilization and decreasing queue size and number. Also, they seek for decreasing the non-value added time generally and decreasing the percentage of rework or junk of specific time-limited processes referred to as Timers. Collaboration between dispatching agents is essential so that they achieve their local and global goals. The collaborative behavior directly affect the survival of timers that involve several stations through processing. The following list shows the component of a dispatching agent's state for the waiting lots within its queue:

- Timer information: within a timer, timer's start, or not
- Timer limit: time until timer's limit
- Lot type
- Product type
- SR
- Number of wafers
- Lot's CTX

In addition to the waiting lots information, there are global details that are included in the state as well that are:

- Numbers of waiting lots in the other stations

- Each equipment state: up/down
- Number of lots arriving after (1-5) processing steps for each type of product

The reward function for a dispatching agent is also related to its station and the general performance of the whole system. The reward function at time t , shown by Eqn. 8.2, contains the following:

- Average CTX for recently produced lots of all types
- Average waiting time and number of lots in the agent's queue
- Average utilization of the agent station's equipment
- Timers processing rating; representing the capability of the agent to process timers without exceeding their time limits
- Negative reward for exceeding timers limits

$$r_{Disp,t} = \frac{CTX_{base}}{CTX_{MA}} + \frac{TQ_{wait,base}}{TQ_{wait,MA}} + \frac{NQ_{wait,base}}{NQ_{wait,MA}} + Util\%_{MA} + \frac{TimRating\%_{base}}{TimRating\%} - TimReward_{exceed} \quad (8.2)$$

The $TQ_{wait,MA}$ and $NQ_{wait,MA}$, define the waiting time and number of lots in agent's queue respectively. The last two terms are related to processing of timers. $TimRating\%$ measures the ratio of wafers with exceeded timers to the total number of wafers within timers that were processed by this agent within their timer's steps. For $TimReward_{exceed}$, it is an instant reward which is assigned upon exceeding a timer's limit, with different value, to agents that are involved in that timer's steps. Because not all agents cause exceeding the timer equally, the assigned value for each agent differs accordingly. This provides better feedback for agents than assignment of a single value. Also, it prevents slow learning and trivial actions by agents such as not starting timers at all to avoid timer's exceeding. More details about the dispatching agents can be found in our previous work [1].

8.5.4 Challenging agents: customers

Customers represent the challenging agents, or opponents, that learn to set due dates creating job orders' tardiness. The amount of job orders is fixed so that the agents only select due

dates. As in case of managing agents, customer agents follow the concept of decision-making decentralization. There is a separate customer agent for each product type. Accordingly, they have separate DQN and replay memories implying independent training.

Implementation of the self-play concept for customer agents requires applying a bounding mechanism for their behavior. Such mechanism is needed to prevent selecting actions that create maximum possible tardiness such as selecting only the tightest available due dates. As discussed in Section 8.4.2, such actions do not necessarily give space for enough training for the managing agents. The requirement from customer agents is to challenge the system adaptively. To solve this problem, customer agents are rewarded based on achieving a certain range of job orders tardiness. This range is set as a certain ratio of job orders' available time.

Along with the reward for achieving such adaptive challenging due dates, customer agents need representative state information. Hence, each agent perceives its product type production progress with respect to the rest of the system including other product types. The state information for customer agents contains the following components:

- Average time in queue for equipment stations
- Ratio of WIP for each product type to the total WIP
- Completion% of process flow steps for the WIP of each product's type

Figure 8.3 shows the reward value assigned for customer agents upon job order completion. The value corresponds to the ratio of job completion time to the job available time set by the customer agent. The job order completion is advanced or tardy if this ratio is less or more than 1 respectively. As shown through the shaded area, the highest reward value is assigned for the specific range between 1 and 1.4. This is the level of orders completion tardiness that customer agents seek for achieving.

Through customers' due dates selection, the calculated SR for the WIP keeps changing more frequently than in other system simulation with no customer agents as in [1]. In our previous work, the order quantities and due dates were set over longer periods and without variation through the simulation. This gave the managing agents more chances to adjust their performance with relatively more stable SR for the WIP. Through introducing customer agents and adopting a self-play training scheme, we expose the managing agents to challenges that they would not face with fixed assigned job orders. The varying SR affects managing agents directly as it is part of their state information. This trains the managing agents for challenging situations that adapt to their level of training. Hence, creating a progressive training for all agents.

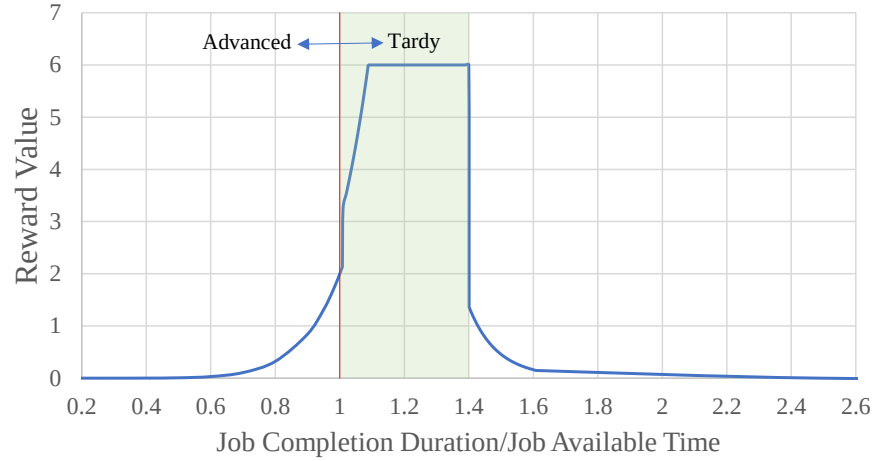


Figure 8.3 Customer agents reward value with respect to job completion time

8.5.5 Conflicting agents: operators and equipment interrupting agents

There are two conflicting agents in the simulation. There is an interrupting agent for operators and another one for equipment. Both select and interrupt the availability of the operators/equipment. The more impact on system's performance due to the interruptions, the higher reward these agents receive. Thus, they learn to select the operator/equipment that is expected to significantly affects the performance of the system.

For operator interruption agent, it selects an active, idle or busy, operator to interrupt. If the operator is busy, loading or unloading an equipment, the interruption is made after the operator finishes his/her task. This affects operator's availability for loading equipment and unloading the equipment that he/she loaded before the interruption if any. The process of interruption is repeated with an interarrival time following an exponential distribution with a mean of 1 hour of simulation time. For equipment interruption agent, the objective is the same as for operator interruption agent. However it interrupts busy, processing, equipment which in turns prolongs its unloading time, and accordingly its step cycle time. Thus affecting the operator's availability who is responsible for unloading the interrupted equipment. The interruption delay is set to be calculated as 20% of the lot's actual elapsed time within the equipment station including the queue time as well.

The state for the conflicting, interruption, agents includes general information about the whole system and operators or equipment activity. The following shows the state information for operator interruption agent:

- Average number and time in queue for equipment stations

- Average CTX, total time in system/ total value-added time, of all products' types.
- Completion% of process flow steps for the WIP of each product's type
- All operators state: active/inactive

For equipment interruption agent, the state differs only in the last component. The equipment state: up/failed is passed instead of operators state. This state includes key information for the agents to be able to conflict the system generally including the managing agents as well.

The interruption agents are rewarded for affecting the system's performance. Eqn. 8.3 shows the reward function for the interruption agents. It is shown that the 3 components in this reward function are related to non-value added time, stations utilization, and production throughput.

$$r_{Interrupt,t} = \frac{CTX_{MA}}{CTX_{base}} + \frac{Util\%_{base}}{Util\%_{overall,MA}} + \frac{TH_{base}}{TH_{MA}} \quad (8.3)$$

8.6 Results and discussion

We applied the proposed methodology to a real semiconductor manufacturing system use case through DES as in our previous work in [1]. In our previous work, we employed only dispatching agents for equipment stations. The results showed a significant reduction in the CTX, the timers' rework and junk percentage, and improvement of the throughput. In this work, we extend the employment of the RL agents to include order release for each product, interruption agents for system resources, and challenging agents representing customers for each product type. This puts the managing agents' objectives in conflict with interruption agents and challenging agents. As mentioned in 8.4.2, this scheme mimics the self-play concept which is applied to improve the learnt policies of RL agents in computer games. Additionally, it mimics real systems challenges through training. Thus, such managing agents are trained for challenging scenarios such as human operator associated variabilities and varying customer demand rates.

The simulation includes two different product types denoted as P1 and P2. They have intersecting process flows that share same equipment stations that have a total number of 25 stations. There is a challenging agent, customer, for each product type. As mentioned in 8.5.4, they set due dates for each job order which has a fixed quantity of 50 wafers. Due to recipes' yield and junk percentages, the 50 wafers are not necessarily provided by two lots as some wafers of the lot get scrapped through the process flow.

Engineering-to-order is needed due to the customer-centric scheme or the required production flexibility. This makes operators subject to interruptions that are required for supervision by their responsible. Thus, it affects their availability and equipment availability as well if unloading is delayed.

8.6.1 DQN configuration for agents

Table 8.2 shows the neural network shape for the agents. The exploration probability for the agents is set as 0.15. For the activation function, the ReLU is used. Each agent's input layer size corresponds to the size of its state. The output layer size represents the number of different actions that each agent selects from. The actions for dispatching agents represents the rank in queue of each lot among the observed waiting lots.

8.6.2 Training parameters

Table 8.1 shows the training parameters for each agent's DQN. Each agent has its own network with its parameters, and its own replay memory. Thus, each agent learns its own policy. This adopts the concept of decision-making decentralization that is considered among strategies for supporting production flexibility.

8.6.3 Actions

Dispatching agents only consider the first 20 waiting lots. For order release agents, their action represents a number in the range [0-20]. This selected number is converted into the Time Till Next Release, $TTNR$, as shown by Eqn. 8.4:

$$TTNR = 80 + 16 \cdot action(min) \quad (8.4)$$

Thus, $TTNR$ ranges from 80 minutes up to 400 minutes with a step of 16 minutes. For interrupting agents, the selection is performed for an active operator or a busy equipment. As there are different working shifts for operators and some equipment may not be busy, we use an action-masking layer according to the total number of busy equipment or active operators at the time of action selection. This prevents the agent from selecting a non-applicable action such as interrupting an inactive operator.

For challenging agents, customers, the actions are due date or time for the next job order to be fulfilled. Table 8.3 shows the corresponding relative due time for each action calculated

Table 8.1 Agents' learning parameters

Parameter	Agents				
	Dispatching	Order release	Customer	Equipment interruption	Operator interruption
Replay memory size	20000	1000	500	2000	2000
Minimum memory size	1000	200	200	200	200
Learning batch size	1000	150	120	200	200
Learning iteration	40	10	4	40	40
Replacement iteration	320	100	120	200	200
Learning rate	0.005	0.002	0.008	0.005	0.005
Discount factor	0.85	0.90	0.01	0.85	0.85

from the action taking instant. Customers set these times when a new job order starts. Thus, this is the available time of that job order at which the order should be fulfilled to avoid tardiness.

8.6.4 Results

We tested the managing agents performance while tackling the effect of their opponents. This involves the resources-availability disturbance caused by the interruption agents, and SR-variability caused by the dynamic demand of the customer agents. To detect improvement of the learnt policies through the applied self-play scheme, we compared some KPIs of the same system in two cases. The first case includes only RL dispatching agents as presented in our previous works [1]. The second case applies RL for all agents: managing, conflicting, and challenging agents.

Figures 8.4 and 8.5 show the lots release times in the two cases. In the first case related to our previous works, the order release was performed through a strategy applying an interarrival time distribution assigned for each product type. As compared to the second case where release time is managed through order release agents, and through Fig. 8.6, it can be shown that the total number of all products at any time is similar in both cases. The difference for each product type is caused by the policy of each order release agent. These policies adjust release times according to the WIP state, and the SR which is varying due to the actions of the challenging customer agents. From these figures, we conclude that the system receives similar total quantities of released lots in both cases.

As discussed in Section 8.5.4, customer agents set due dates, or times, for job orders of a fixed quantities of 50 wafers. The distribution of customer agents' set available times for each job order is shown in Fig. 8.7. This figure shows the variability of the selected available time for each job order. This changes the state of the WIP as the SR changes with each new job

Table 8.2 DQNs' parameters for all agents

Parameter	Dispatching	Order release	Customer	Equipment interruption	Operator interruption
Input layer size	176	107	47	105	114
Output layer size	21	21	9	31	40
Hidden layer 1	140	85	37	84	93
Hidden layer 2	88	53	23	52	57

Table 8.3 Customer agents actions

Action no.	Due in (hrs)
Action 1	16
Action 2	12
Action 3	8
Action 4	7
Action 5	6
Action 6	5
Action 7	4
Action 8	3.5
Action 9	2

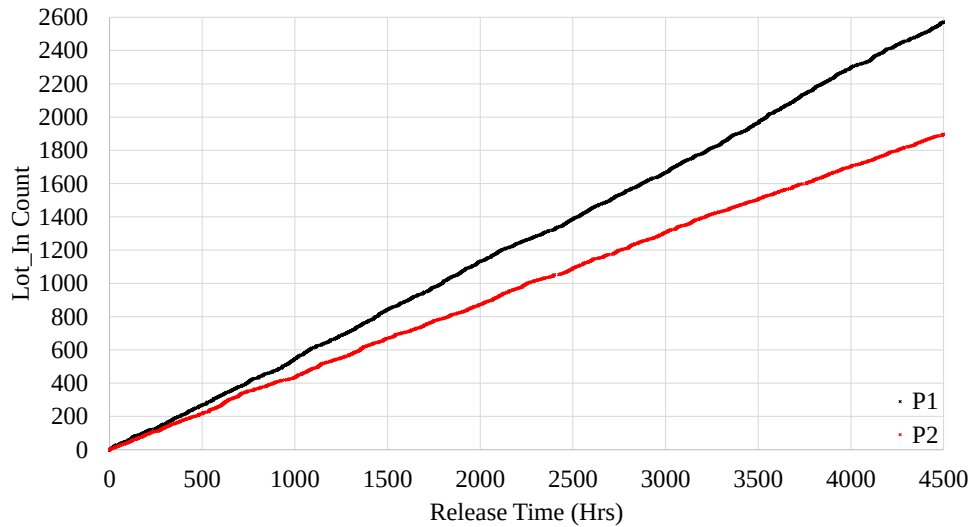


Figure 8.4 Release times for lots in system applying RL for dispatching agents only

order. This adds an extra challenge for the managing agents, order release and dispatching agents, as they need to tackle the effect of this variability.

Figures 8.8 and 8.9 show the CTX in the two cases. As shown in the figures, the system with RL dispatching only agents shows generally lower levels of CTX than the system with conflicting and challenging agents. This is expected as in the first case, there were neither

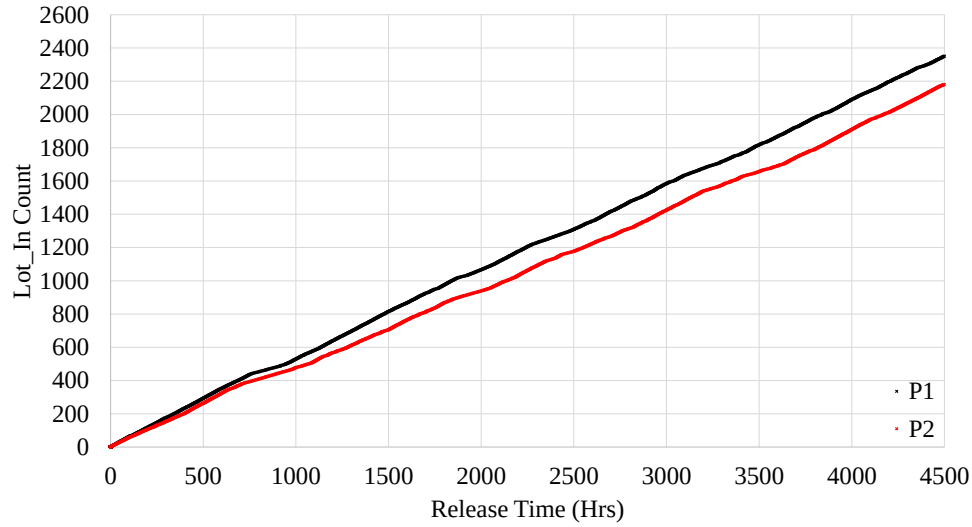


Figure 8.5 Release times for lots in system applying RL for managing, conflicting, and challenging agents

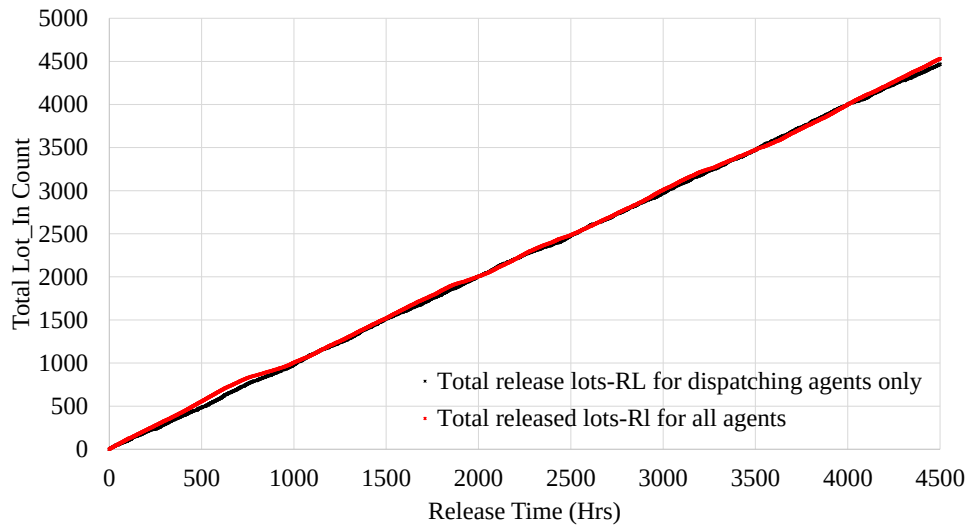


Figure 8.6 Total released lots with only RL dispatching agents and with RL for all agents

interruptions nor dynamic demand rates as in our case. The presence of the conflicting and challenging agents caused extra congestion within the system, hence the CTX levels increased.

Given the extra congestion added in our case due to the conflicting and challenging agents, the percentages of rework and junk for timers are expected to increase as well. However, it is shown in Fig. 8.10-8.11 that the percentages of timers' rework and junk were decreased. This indicates that the dispatching agents reached better policies when exposed to the conflicts

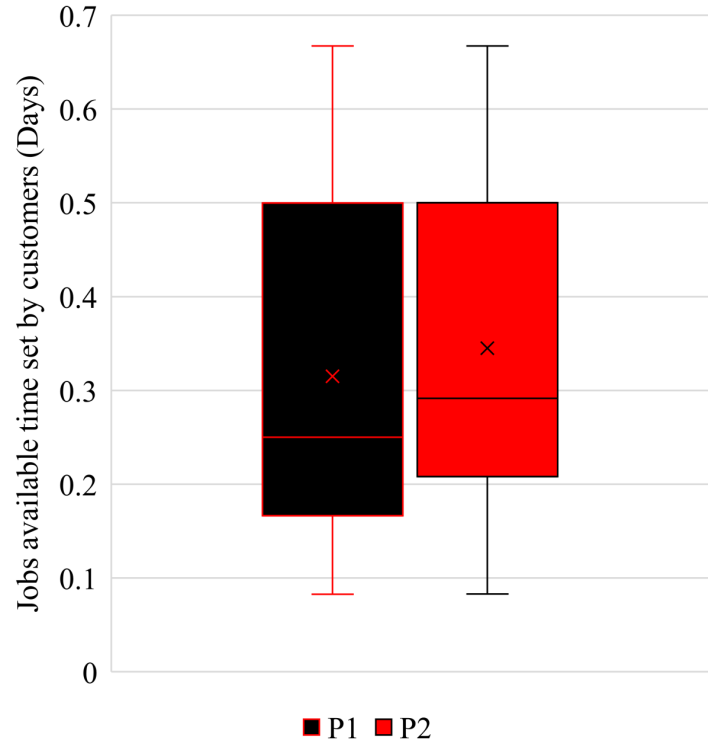


Figure 8.7 Box-plot for the available times for job orders set by customer agents

caused by the presence of the interruption and customer agents.

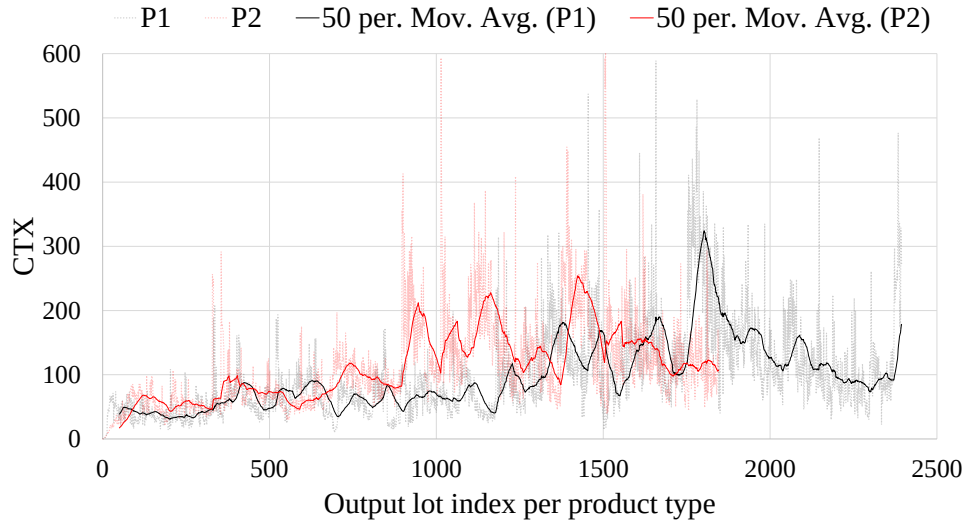


Figure 8.8 CTX in system applying RL for dispatching agents only

Managing agents are challenged by customers to fulfill the job orders without tardiness. We calculated the slack time for each job order upon completion as an absolute value and

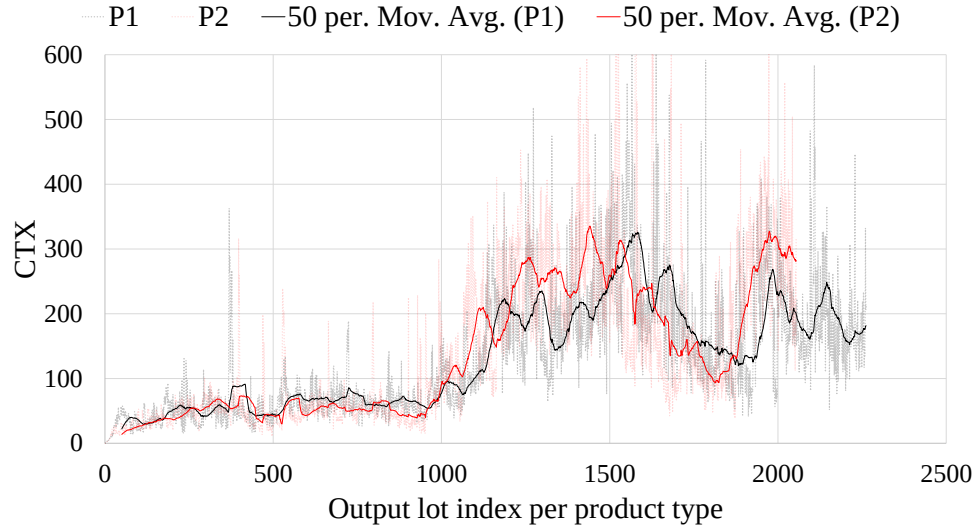


Figure 8.9 CTX in system applying RL for managing, conflicting, and challenging agents

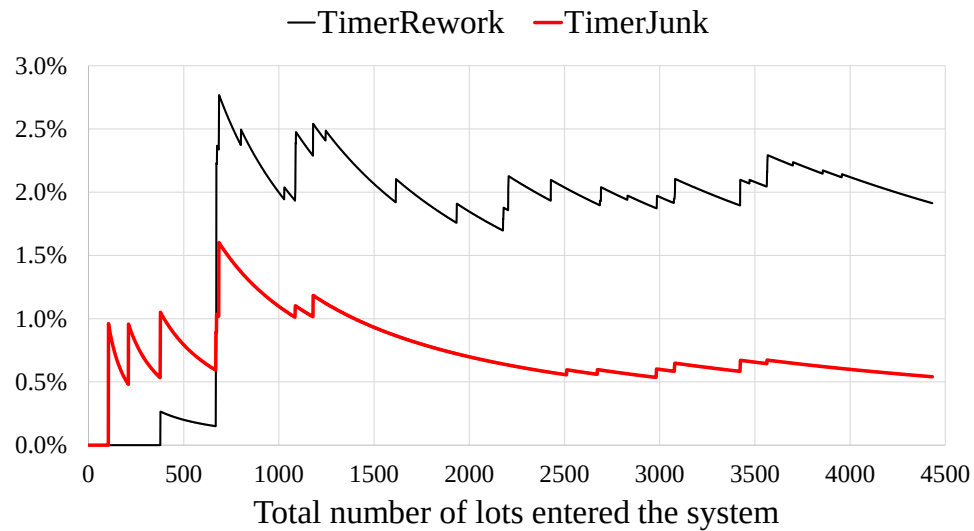


Figure 8.10 Percentages of rework and junk of timers in system applying RL for dispatching agents only

as a ratio to the available time set by the customer agents. Figures 8.12 and 8.13 show the absolute value for the jobs' slack time upon orders completion and their distribution in a box-plot respectively. The figures shows that most of the jobs are fulfilled in advance. Figures 8.14 and 8.15 show the slack time upon job completion as related to the jobs available time. The averages of that ratio are 0.22 and 0.3, and the medians are 0.86 and 0.71 for P1 and P2 respectively. These values show that the managing agents handled the varying SR caused by the dynamic demand of the customer agents.

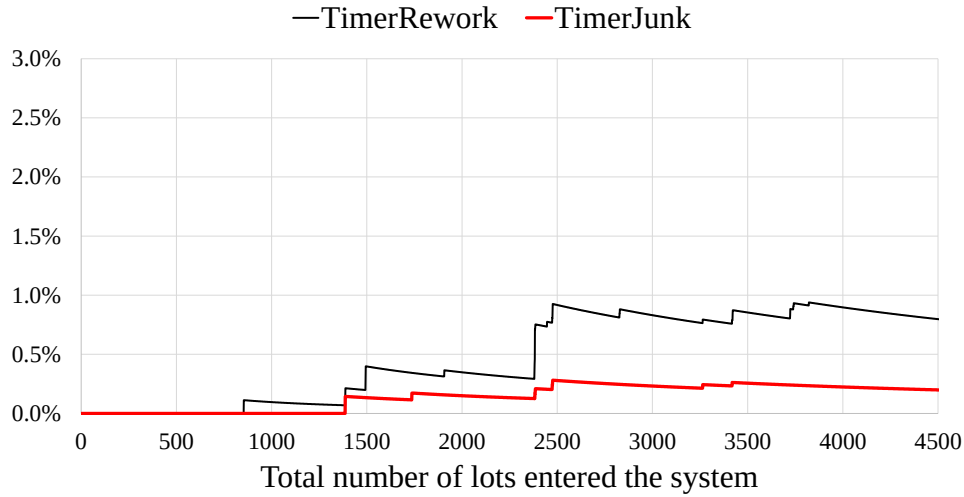


Figure 8.11 Percentages of rework and junk of timers in system applying RL for managing, conflicting, and challenging agents

The percentages of tardy jobs are 11.06% and 12.19% for P1 and P2 respectively. There is a specific range of tardiness that customer agents seek for achieving according to their reward function as shown in Fig. 8.3. According to that figure, this range is bounded by a maximum of 40% tardiness of jobs' available time. The percentages of tardy jobs that lie within this range are 3% and 6.1% for P1 and P2 respectively. These percentages indicate that the variability caused by customer agents for the demand rates are mostly handled by the managing agents.

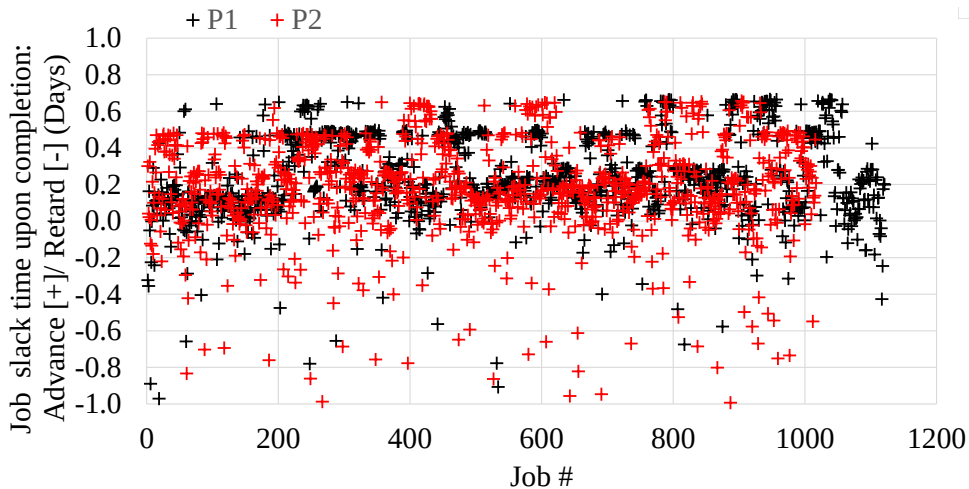


Figure 8.12 Absolute slack time for jobs upon completion

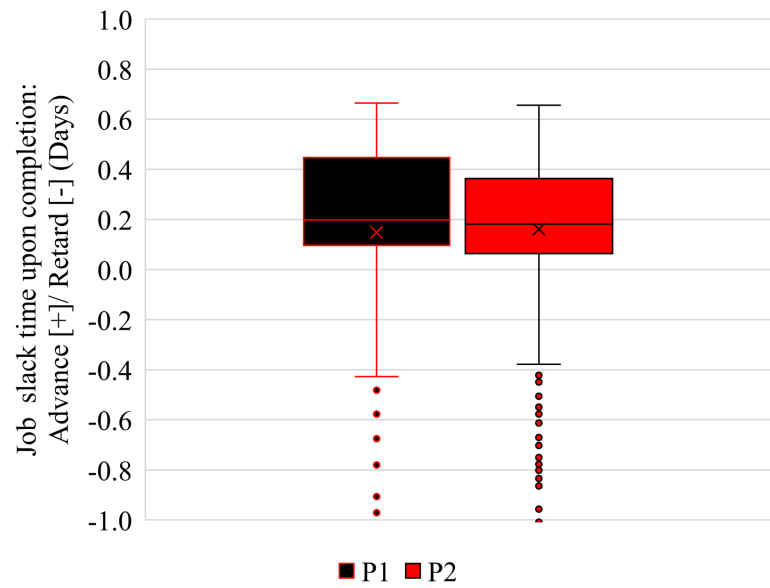


Figure 8.13 Box-plot for absolute slack time for jobs upon completion

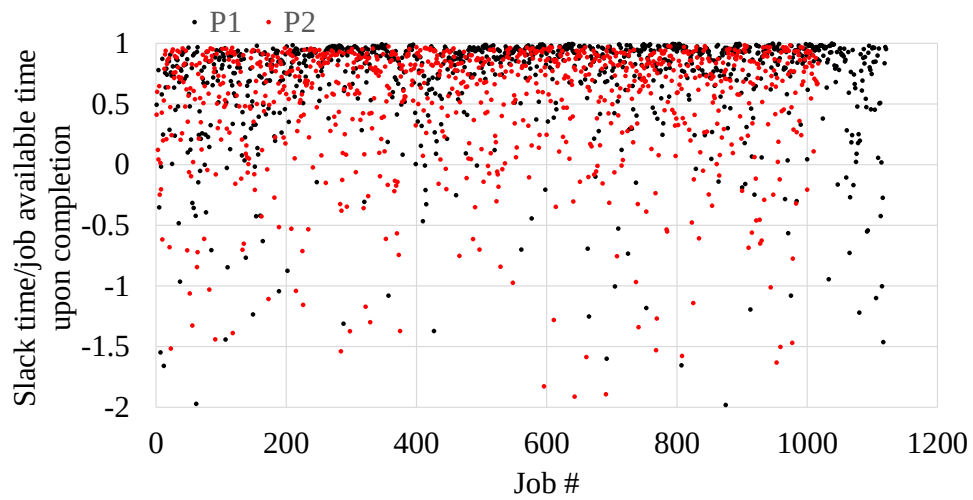


Figure 8.14 Slack to available time ratio for jobs upon completion

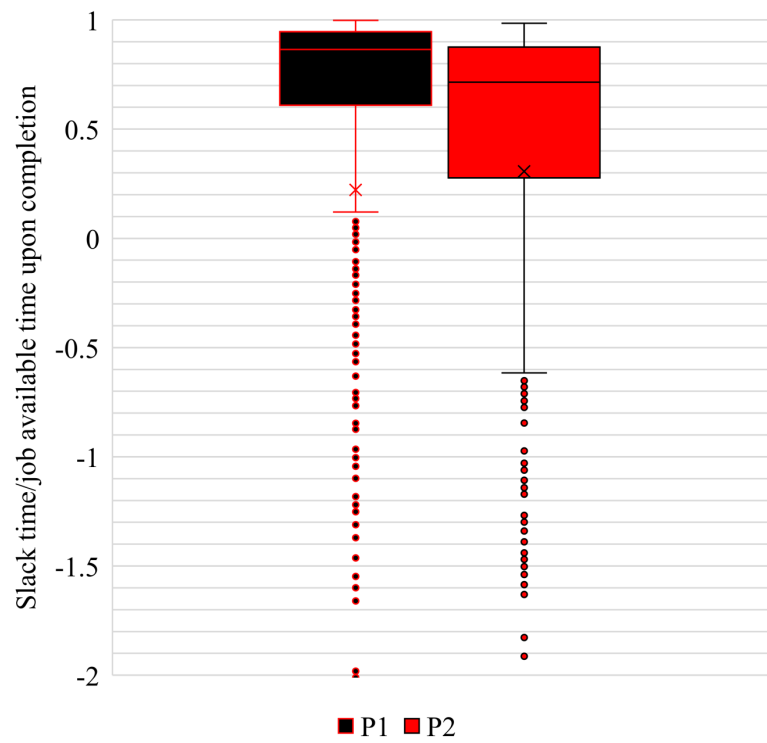


Figure 8.15 Box-plot for slack to available time ratio for jobs upon completion

8.7 Conclusion

In this paper, we introduced an approach for implementing the self-play concept for RL customer agents. They were deployed in a DES model of a manufacturing system in which managing agents were responsible for order release and dispatching decisions. The self-play approach is motivated by the need for improving the managing agents' learnt policies without increasing the training time. The customer agents challenged the managing agents through a variable demand. They sought for achieving a certain range of tardiness for the job orders completion. That range of tardiness was set to challenge the managing agents. This was done to avoid confronting the managing agents with excessively hard situations that affect or hinder learning suitable policies. In addition, the set range of tardiness prevented the customer agents from trivially selecting the tightest available due dates if the required tardiness was unbounded. We included other conflicting agents that were responsible for creating interruptions for system resources, equipment and operators. The selection was for the system resource that significantly impacts the general performance.

Through this scheme, the managing agents were encouraged to actively improve their policies to overcome the variability caused by the customer agents. Also, they were forced to handle the impacts of the interruptions that the conflicting agents created. The difficulties created by both challenging and conflicting agents were actively opposing the managing agents as all agents are learning together. Unlike relying on randomly assigned interruptions and due dates, there was no need for prolonging the training time to improve the managing agents policies. This prevented the other learning problems that appears with prolonged training such as performance plateaus.

The customer agents assigned variable due dates for their job orders. However, the results showed that job orders were mostly fulfilled with no tardiness. We compared the percentages of rework and junk in our case to the same system with dispatching agents only and no interruptions or variable demand. The comparison showed that the system with conflicting and challenging agents decreased the rework and junk percentages by more than half. This indicates that the dispatching agents, having the same DQN configurations, learnt better policies when the other agents were present. This was achieved even though the system was more congested due to the conflicting agents impact.

Acknowledgement

This work is funded through the Mitacs Accelerate program, Grant No. IT01632, in partnership with Teledyne DALSA Semiconductor, a business unit of Teledyne Digital Imaging,

Inc. The authors would like to thank the industrial and information technology teams at Teledyne DALSA semiconductor for their great support and collaboration.

CHAPTER 9 GENERAL DISCUSSION

9.1 Summary

In this research, Reinforcement Learning (RL) is applied to support decision-making in complex production systems. The RL was applied through the Deep-Q-Network (DQN) algorithm for multiple agents. The use of multiple agents was adopted to achieve decision-making decentralization, which supports the required flexibility in production systems. Discrete-Event Simulation (DES) was employed to provide a model representing the production system. The DES model provided the training environment for the agents. We applied our approach through a use case for a real semiconductor manufacturing system.

The objectives of this research are categorized as follows:

Objective A: Develop a DES model considering Digital Twin (DT) concepts.

A Hybrid discrete-Event Simulation was developed for the manufacturing system use case by applying the concepts of Data-driven and Agent-based modeling. The focus was on structuring the input data that were acquired by the model throughout its execution. Data-driven model building was adopted to ensure flexibility to accommodate the behavior of the use case as a complex system. Agent-based modeling was applied to provide the necessary functionalities for agents, allowing decisions made by RL agents to be applied during model execution and to provide the training environment for agents.

Objective B: Apply RL for decision-making and achieve cooperation among the managing agents.

Managing agents were created for dispatching and order release decisions, with the adoption of decision-making decentralization and inspired by the holonic manufacturing concepts. The training was accomplished through the simulated environment, and the reward is calculated based on the production performance. Different KPIs, such as cycle time, resource utilization, and waste reduction, were utilized as performance metrics. The formulation of state and reward aimed to encourage cooperation and discourage a narrow focus on localized objectives.

Objective C: Develop a self-play approach to employ real systems variabilities for improving the learnt policies of the managing agents.

Real systems variabilities, including the presence of human operators and resource interruptions, were taken into account in the research. Additionally, demand variation was addressed. To introduce these variabilities within the simulation, conflicting and challenging RL agents

were incorporated. This approach drew inspiration from the self-play concept commonly applied in computer games. Consequently, both managing agents and other agents were incentivized to explore different strategies to overcome the effects of one another. Through this approach, agents acquired the ability to handle diverse and atypical situations encountered in real systems. Importantly, this approach enhanced the learnt policies of managing agents without the need to extend the training time.

In the first part of this thesis, in Chapter 4, a DES modelling logic was established for a semiconductor use case, which included specific process flows for each product type, as well as preprocessing and postprocessing steps. The modelling logic was based on hybrid simulation applying both data-driven DES and agent-based modeling, which allowed for flexibility and decision-making capabilities within the agents. In Chapter 5, the DES model was further developed using DT concepts to accommodate changes and real-time decision-making. The model included two additional features, model reconfiguration and state initiation.

In Chapter 6, RL is used to replace heuristics for dispatching agents, resulting in improved system performance. The DQN approach was compared to some benchmark heuristics-based dispatching method and was found to improve system performance, particularly in terms of throughput, non-value added time reduction, and timers' exceeding reduction. The results were achieved by properly defining the system's state information and reward function to facilitate cooperation among agents. A well-defined reward function helped to prevent the agents from making trivial decisions.

Chapter 7 introduced interruptions such as human operators and equipment malfunctions into the simulation. This strategy improved the policies of managing agents responsible for dispatching in a production system by introducing RL conflicting agents following the concept of self-play. These conflicting agents were designed to replace random interruptions. The performance was compared to traditional heuristics and RL dispatching agents only with no interruptions, presented in Chapter 6, and the results showed an improvement in the dispatching agents' policies. The improvement was achieved within the same training time as managing agents with no interruptions. This makes the strategy more applicable to real systems considering the human operators' presence. Results showed that this approach decreased rework and junk percentages by more than half compared to a system with dispatching agents only and no interruptions.

Chapter 8 extended this concept to include customer agents challenging the system with varying demand rates, resulting in better policy learning for dispatching agents. The study demonstrated that the managing agents handled variable demand rates and changing SRs while still fulfilling job orders with minimal tardiness.

The research conclusions are as follows:

- RL, through DQN, was applied for managing agents to support decision-making for dispatching and order release tasks. Decision-making decentralization was approached through employing a separate DQN and replay memory for each agent so that they are trained individually.
- State information selection and reward function formulation contributed to cooperation achievement among the managing agents and prevented them from focusing on local objectives. This was detected for the reward function in cases of closely linked actions such as handling multiple processing steps with time constraints.
- The self-play concept was applied to develop conflicting agent, representing system resources interruption, to maximize the impact of these disturbances within the limited training time. This improved the managing agents' policies within the same training period. In addition, it trained the agents for human presence in real system and its associated variabilities.
- A game-like self-play scheme between the production system and its customers was introduced. This considered the demand variability that customers create in reality, and trained for this type of variability. It showed the managing agents' capability to handle variable demand through the proposed training scheme.

The contributions of this research are as follows:

In Article 3, a comprehensive approach for applying RL-based decision-making in semiconductor manufacturing systems was introduced. This included hybrid DES and agent-based modelling and DQN for dispatching agents. The state information and reward function were formulated to support cooperative behavior among agents. This was done considering specific real production challenges that require direct cooperation among agents to tackle.

In Article 4, the real system sources of variability caused by resources interruptions were considered during training. The significance of these events were increased, within the limited training time, through the proposed self-play scheme for conflicting agents. This trained the managing agents to handle less frequent events, mainly system resources interruptions, and to employ these events to improve the learnt policies without the need for prolonged training period.

In Article 5, a self-play scheme between the system and customers was introduced. The customers were represented by challenging agents that vary their demand rate during the

simulation. This trains the managing agents for handling variable customer demands in an adaptive way according to their learnt policies.

9.2 Future Research

There are some limitations for applying

Given the improved performance achieved through applying RL-based decision making and self-play concepts, the following points are our future perspectives for extending the proposed approach's scope of application:

- As the actions taken by the agents are not explained, the replay memory for each agent can be analyzed to provide an explanation for the general policy of the agent. This can be accomplished by clustering the replay memory on many known heuristics, resulting in an inference about each agent's policy.
- The self-play scheme can be extended in its application to include pricing decisions for job orders . An agent, representing the manufacturer's management, can be introduced and trained through a game-like scheme against customer agents. This would be beneficial for manufacturers as such a decision considers many aspects, including system states and delivery times required by customers.

CHAPTER 10 CONCLUSION AND RECOMMENDATIONS

In this research, RL was applied for decision-making in complex production systems. A use case for a real semiconductor manufacturing system was utilized to account for the challenging aspects that are found in real systems. In the use case, these aspects include production complexities due to the discrete and process manufacturing characteristics of semiconductor industry. In addition, there is a challenge experienced through decision-making due to the associated variability to the human operator presence. The demand variability caused by the move towards customer-centric production adds more challenges to management of these production systems.

Complex production systems, such as semiconductor industry, often involve numerous sources of challenges that must be considered when applying RL techniques. In semiconductor industry, wafers, usually in the form of lots of 25 wafers, are processed together. They are not allowed to mix with wafers from other lots. Lots undergo splitting events in which they are divided into several parts that are processed individually and later rejoined. Equipment must be loaded and unloaded by operators whose availability is not fixed. Further, some successive processing steps must pass through multiple equipment while respecting certain time constraints.

All of these aspects made the application of RL challenging, especially in a multi-agent environment where cooperation is required. It was crucial to formulate state information and reward functions properly to ensure that the agents navigate the complex production system. Merely setting fixed values for agent rewards in specific events was insufficient, as the system's behavior is highly dynamic and subject to many sources of variability.

The main objective of this research was to create a decision-making support tool specifically designed for complex production systems. To accomplish this, the research focused on simulating the real system to represent its complex behavior. Additionally, the research introduced RL-based agents to embed intelligence into the decision-making process. Managing agents were trained to make appropriate decisions by engaging with the simulated environment, which served as the training environment. These managing agents were responsible for making decisions related to dispatching and order release tasks. To promote cooperation among agents, the state information and reward functions were formulated accordingly considering self and common states and objectives.

Drawing inspiration from the self-play concept, the managing agents were challenged by conflicting and challenging agents. This setup was intended to enhance the learnt policies

of the managing agents, accounting for the presence of human operators, typical resource interruptions, and demand variability commonly observed in real systems. By employing this training strategy, the research aimed to avoid the need for prolonged training periods.

To fulfill the research objectives, decentralized decision-making was achieved through the employment of multiple RL agents. The applied RL technique was DQN, in which each agent was trained individually through its own replay memory. To provide the training environment for the agents, a hybrid DES model, through DES modeling and agent-based modeling, was developed for a use case of a real semiconductor manufacturing system. To improve the learnt policies of the agents, without prolonging the training time, a self-play scheme was applied. This scheme improved the learnt policies of the agents and trained them for different sources of variability that exist in real systems.

The proposed approach includes managing agents, which were responsible for dispatching and order release tasks. There were other agents which were part of the self-play scheme. These agents were conflicting and challenging. Their objectives were to impact the system's KPIs and challenge the managing agents. The conflicting agents were responsible for interrupting system resources, operators and equipment. The challenging agents were representing customers, which vary the demand rate for their job orders for each product type. Through data-driven and agent-based modeling techniques, the hybrid DES modeling was facilitated to represent the complex behavior of the use case system. It provided the functionalities required for implementing RL-based decision-making within the simulation. The state and reward formulations for managing agents were done considering the required cooperative behavior. Our approach outlined the aspects to be considered when selecting state information. The importance of reward function formulation, for supporting cooperative behavior among agents, was shown in consecutive processing steps with time constraints, known in the semiconductor industry as timers. The proposed formulation avoided trivial actions caused by improper reward assignment, such as not starting timers to avoid exceeding the timer's limit. The results showed that the dispatching agents outperformed heuristics, including the hybrid heuristic that is applied in the use case. The self-play scheme was applied to include conflicting agents. This trained the managing agents, dispatching and order release, to consider the presence of human operators in such systems, along with the equipment interruptions that take place during processing. Further, it improved the learnt policies of the managing agents within the same training time. Finally, a game-like self-play scheme was introduced in which customers and the managing agents are opponents where customers vary their demand rate. This trained the managing agents for such variability and showed their readiness to handle it. This was evident in the results, as most job orders were fulfilled without tardiness.

The application of RL-based decision-making in production systems is a promising approach. Considering the self-play scheme applied in a production system context, decision-making agents can be trained for challenging events that are relatively less frequent and hard to be trained for. This approach trains them efficiently for such events. It actively forces the agents to improve their learnt policies, with no need to prolong training periods. I hope our approach will help to bring the application of RL closer to reality.

REFERENCES

- [1] A. H. Sakr *et al.*, “Simulation and deep reinforcement learning for adaptive dispatching in semiconductor manufacturing systems,” *Journal of Intelligent Manufacturing*, pp. 1–14, 2021.
- [2] E. Negria, L. Fumagalli, and M. Macchi, “A review of the roles of Digital Twin in CPS-based production systems,” in *International Conference on Flexible Automation and Intelligent Manufacturing*, Modena, Italy, 2017.
- [3] J. Lee, B. Bagheri, and H.-A. Kao, “A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems,” *Manufacturing letters*, vol. 3, pp. 18–23, 2015.
- [4] M. Javaid *et al.*, “Enabling flexible manufacturing system (fms) through the applications of industry 4.0 technologies,” *Internet of Things and Cyber-Physical Systems*, vol. 2, pp. 49–62, 2022.
- [5] “Reinforcement learning for adaptive order dispatching in the semiconductor industry,” *CIRP Annals*, vol. 67, no. 1, pp. 511 – 514, 2018.
- [6] C. Zhuang, J. Liu, and H. Xiong, “Digital twin-based smart production management and control framework for the complex product assembly shop-floor,” *The International Journal of Advanced Manufacturing*, vol. 96, pp. 1149–1163, 2018.
- [7] “Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination,” *Computer Networks*, vol. 101, pp. 158 – 168, 2016, industrial Technologies and Applications for the Internet of Things.
- [8] M. Ismail, N. A. Mostafa, and A. El-assal, “Quality monitoring in multistage manufacturing systems by using machine learning techniques,” *Journal of Intelligent Manufacturing*, pp. 1–16, 2021.
- [9] “Automatic design of scheduling rules for complex manufacturing systems by multi-objective simulation-based optimization,” *CIRP Annals*, vol. 65, no. 1, pp. 433 – 436, 2016.
- [10] T. Pencarelli, “The digital revolution in the travel and tourism industry,” *Information Technology & Tourism*, vol. 22, no. 3, pp. 455–476, 2020.

- [11] J. Zhang *et al.*, “Review of job shop scheduling research and its new perspectives under industry 4.0,” *Journal of Intelligent Manufacturing*, vol. 30, no. 4, pp. 1809–1830, 2019.
- [12] K. Ashton *et al.*, “That ‘internet of things’ thing,” *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.
- [13] J. Lee, H.-A. Kao, and S. Yang, “Service innovation and smart analytics for Industry 4.0 and big data environment,” *Procedia CIRP*, vol. 16, pp. 3–8, 2014.
- [14] A. Kuhnle *et al.*, “Design, implementation and evaluation of reinforcement learning for an adaptive order dispatching in job shop manufacturing systems,” *Procedia CIRP*, vol. 81, pp. 234–239, 2019.
- [15] J. Vachálek *et al.*, “The digital twin of an industrial production line within the industry 4.0 concept,” in *2017 21st International Conference on Process Control (PC)*, 2017, pp. 258–262.
- [16] T. H.-J. Uhlemann, C. Lehmann, and R. Steinhilper, “The digital twin: Realizing the cyber-physical production system for industry 4.0,” *Procedia Cirp*, vol. 61, pp. 335–340, 2017.
- [17] S. Miller and D. Pegden, “Introduction to manufacturing simulation,” in *2000 Winter Simulation Conference Proceedings (Cat. No. 00CH37165)*, vol. 1. IEEE, 2000, pp. 63–66.
- [18] J. W. Fowler, L. Mönch, and T. Ponsignon, “Discrete-event simulation for semiconductor wafer fabrication facilities: A tutorial,” *International Journal of Industrial Engineering*, vol. 22, no. 5, 2015.
- [19] G. S. Fishman, *Discrete-event simulation: modeling, programming, and analysis*. Springer Science & Business Media, 2013.
- [20] L. Monostori, J. Váncza, and S. R. Kumara, “Agent-based systems for manufacturing,” *CIRP annals*, vol. 55, no. 2, pp. 697–720, 2006.
- [21] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [22] B. Waschneck *et al.*, “Deep reinforcement learning for semiconductor production scheduling,” in *2018 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 2018, pp. 301–306.

- [23] C. Wagner *et al.*, “The role of the industry 4.0 asset administration shell and the digital twin during the life cycle of a plant,” in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–8.
- [24] A. G. Frank, L. S. Dalenogare, and N. F. Ayala, “Industry 4.0 technologies: Implementation patterns in manufacturing companies,” *International Journal of Production Economics*, vol. 210, pp. 15 – 26, 2019.
- [25] D. Silver *et al.*, “Mastering chess and shogi by self-play with a general reinforcement learning algorithm,” *arXiv preprint arXiv:1712.01815*, 2017.
- [26] D. Silve *et al.*, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
- [27] N. Prajapat and A. Tiwari, “A review of assembly optimisation applications using discrete event simulation,” *International Journal of Computer Integrated Manufacturing*, vol. 30, no. 2-3, pp. 215–228, 2017.
- [28] S. M. Jeon and G. Kim, “A survey of simulation modeling techniques in production planning and control (ppc),” *Production Planning & Control*, vol. 27, no. 5, pp. 360–377, 2016.
- [29] C. Reményi and S. Staudacher, “Systematic simulation based approach for the identification and implementation of a scheduling rule in the aircraft engine maintenance,” *International Journal of Production Economics*, vol. 147, pp. 94 – 107, 2014.
- [30] U. Ghani, R. P. Monfared, and R. Harrison, “Energy optimisation in manufacturing systems using virtual engineering-driven discrete event simulation,” *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 226, no. 11, pp. 1914–1929, 2012.
- [31] T. McDonald, E. Akenb, and K. Ellib, “Utilizing simulation to evaluate production line performance under varying demand conditions,” *International Journal of Industrial Engineering Computations*, vol. 3, no. 1, pp. 3–14, 2012.
- [32] P. Korytkowski, T. Wiśniewski, and S. Rymaszewski, “An evolutionary simulation-based optimization approach for dispatching scheduling,” *Simulation Modelling Practice and Theory*, vol. 35, pp. 69–85, 2013.
- [33] G. P. Cerna *et al.*, “Using discrete event simulation to muck development planning in underground mining,” *Procedia Computer Science*, vol. 220, pp. 916–921, 2023.

- [34] C. Strauss *et al.*, “Optimizing emergency medical service structures using a rule-based discrete event simulation—a practitioner’s point of view,” *International Journal of Environmental Research and Public Health*, vol. 18, no. 5, p. 2649, 2021.
- [35] T.-Y. Hong and C.-F. Chien, “A simulation-based dynamic scheduling and dispatching system with multi-criteria performance evaluation for industry 3.5 and an empirical study for sustainable tft-lcd array manufacturing,” *International Journal of Production Research*, vol. 58, no. 24, pp. 7531–7547, 2020.
- [36] S. Lang *et al.*, “Modeling production scheduling problems as reinforcement learning environments based on discrete-event simulation and openai gym,” *IFAC-PapersOnLine*, vol. 54, no. 1, pp. 793–798, 2021.
- [37] A. Greasley and J. S. Edwards, “Enhancing discrete-event simulation with big data analytics: A review,” *Journal of the Operational Research Society*, vol. 72, no. 2, pp. 247–267, 2021.
- [38] H. Zhang *et al.*, “A digital twin-based approach for designing and multi-objective optimization of hollow glass production line,” *IEEE Access*, vol. 5, pp. 26 901–26 911, 2017.
- [39] Y. Cai *et al.*, “Sensor data and information fusion to construct digital-twins virtual machine tools for cyber-physical manufacturing,” *Procedia Manufacturing*, vol. 10, pp. 1031–1042, 2017.
- [40] C. H. d. Santos *et al.*, “Use of simulation in the industry 4.0 context: Creation of a digital twin to optimise decision making on non-automated process,” *Journal of Simulation*, vol. 16, no. 3, pp. 284–297, 2022.
- [41] S. Reed, M. Löfstrand, and J. Andrews, “Modelling cycle for simulation digital twins,” *Manufacturing Letters*, vol. 28, pp. 54–58, 2021.
- [42] H. Qiu *et al.*, “Evolutionary digital twin model with an agent-based discrete-event simulation method,” *Applied Intelligence*, vol. 53, no. 6, pp. 6178–6194, 2023.
- [43] K. Mykoniatis and G. A. Harris, “A digital twin emulator of a modular production system using a data-driven hybrid modeling and simulation approach,” *Journal of Intelligent Manufacturing*, pp. 1–13, 2021.
- [44] G. Lugaresi and A. Matta, “Automated manufacturing system discovery and digital twin generation,” *Journal of Manufacturing Systems*, vol. 59, pp. 51–66, 2021.

- [45] E. Hozdić and I. Makovec, “Evolution of the human role in manufacturing systems: On the route from digitalization and cybernation to cognitization,” *Applied System Innovation*, vol. 6, no. 2, p. 49, 2023.
- [46] H. ElMaraghy and W. ElMaraghy, “Adaptive cognitive manufacturing system (acms)—a new paradigm,” *International Journal of Production Research*, vol. 60, no. 24, pp. 7436–7449, 2022.
- [47] A. F. Buckhorst, L. Grahm, and R. H. Schmitt, “Decentralized holonic control system model for line-less mobile assembly systems,” *Robotics and Computer-Integrated Manufacturing*, vol. 75, p. 102301, 2022.
- [48] D. A. Rossit, F. Tohmé, and M. Frutos, “A data-driven scheduling approach to smart manufacturing,” *Journal of Industrial Information Integration*, vol. 15, pp. 69–79, 2019.
- [49] W. Derigent, O. Cardin, and D. Trentesaux, “Industry 4.0: contributions of holonic manufacturing control architectures and future challenges,” *Journal of Intelligent Manufacturing*, vol. 32, no. 7, pp. 1797–1818, 2021.
- [50] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, 2015.
- [51] C. Jiang *et al.*, “Machine learning paradigms for next-generation wireless networks,” *IEEE Wireless Communications*, vol. 24, no. 2, pp. 98–105, 2016.
- [52] T. Wuest *et al.*, “Machine learning in manufacturing: advantages, challenges, and applications,” *Production & Manufacturing Research*, vol. 4, no. 1, pp. 23–45, 2016.
- [53] D. Pham and A. Afify, “Machine-learning techniques and their applications in manufacturing,” *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 219, no. 5, pp. 395–412, 2005.
- [54] E. Alpaydin, *Introduction to machine learning*. MIT press, 2009.
- [55] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, “Supervised machine learning: A review of classification techniques,” *Emerging artificial intelligence applications in computer engineering*, vol. 160, pp. 3–24, 2007.
- [56] K. Hansson *et al.*, “Machine learning algorithms in heavy process manufacturing,” *American Journal of Intelligent Systems*, vol. 6, no. 1, pp. 1–13, 2016.

- [57] B. Csáji, L. Monostori, and B. Kádár, “Reinforcement learning in a distributed market-based production control system,” 7 2006.
- [58] H. Oliff and Y. Liu, “Towards industry 4.0 utilizing data-mining techniques: A case study on quality improvement,” *Procedia CIRP*, vol. 63, pp. 167 – 172, 2017, manufacturing Systems 4.0 – Proceedings of the 50th CIRP Conference on Manufacturing Systems.
- [59] D. Ouelhadj and S. Petrovic, “A survey of dynamic scheduling in manufacturing systems,” *Journal of scheduling*, vol. 12, no. 4, pp. 417–431, 2009.
- [60] N. Stricker *et al.*, “Reinforcement learning for adaptive order dispatching in the semiconductor industry,” *CIRP Annals*, vol. 67, no. 1, pp. 511–514, 2018.
- [61] J. Shahrabi, M. A. Adibi, and M. Mahootchi, “A reinforcement learning approach to parameter estimation in dynamic job shop scheduling,” *Computers & Industrial Engineering*, vol. 110, pp. 75–82, 2017.
- [62] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.
- [63] “Application of reinforcement learning for agent-based production scheduling,” *Engineering Applications of Artificial Intelligence*, vol. 18, no. 1, pp. 73 – 82, 2005.
- [64] R. Y. Zhong *et al.*, “Intelligent manufacturing in the context of industry 4.0: a review,” *Engineering*, vol. 3, no. 5, pp. 616–630, 2017.
- [65] A. Kuhnle and G. Lanza, “Application of reinforcement learning in production planning and control of cyber physical production systems,” in *Machine Learning for Cyber Physical Systems*, J. Beyerer, C. Kühnert, and O. Niggemann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 123–132.
- [66] C.-L. Liu, C.-C. Chang, and C.-J. Tseng, “Actor-critic deep reinforcement learning for solving job shop scheduling problems,” *Ieee Access*, vol. 8, pp. 71 752–71 762, 2020.
- [67] C.-C. Lin *et al.*, “Smart manufacturing scheduling with edge computing using multiclass deep q network,” *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4276–4284, 2019.
- [68] T. Altenmüller *et al.*, “Reinforcement learning for an intelligent and autonomous production control of complex job-shops under time constraints,” *Production Engineering*, vol. 14, no. 3, pp. 319–328, 2020.

- [69] A. B. Said *et al.*, “Towards proactive maintenance actions scheduling in the semiconductor industry (si) using bayesian approach,” *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 544–549, 2016.
- [70] “A big data analytics architecture for cleaner manufacturing and maintenance processes of complex products,” *Journal of Cleaner Production*, vol. 142, pp. 626 – 641, 2017, special Volume on Improving natural resource management and human health to ensure sustainable societal development based upon insights gained from working within ‘Big Data Environments’.
- [71] A. M. Law, *Simulation modeling and analysis*. McGraw-Hill New York, 2015, vol. 5.
- [72] J. Leng *et al.*, “Digital twin-driven manufacturing cyber-physical system for parallel controlling of smart workshop,” *Journal of ambient intelligence and humanized computing*, vol. 10, no. 3, pp. 1155–1166, 2019.
- [73] C. Cimino, E. Negri, and L. Fumagalli, “Review of digital twin applications in manufacturing,” *Computers in Industry*, vol. 113, p. 103130, 2019.
- [74] W. de Paula Ferreira, F. Armellini, and L. A. De Santa-Eulalia, “Simulation in industry 4.0: A state-of-the-art review,” *Computers & Industrial Engineering*, p. 106868, 2020.
- [75] P. Goodall, R. Sharpe, and A. West, “A data-driven simulation to support remanufacturing operations,” *Computers in Industry*, vol. 105, pp. 48–60, 2019.
- [76] R. Calegari *et al.*, “Logic-based technologies for multi-agent systems: a systematic literature review,” *Autonomous Agents and Multi-Agent Systems*, 2021.
- [77] I. Kovalenko, D. Tilbury, and K. Barton, “The model-based product agent: A control oriented architecture for intelligent products in multi-agent manufacturing systems,” *Control Engineering Practice*, vol. 86, pp. 105–117, 2019.
- [78] A. AboElHassan, A. Sakr, and S. Yacout, “A framework for digital twin deployment in production systems,” in *Advances in Automotive Production Technology—Theory and Application*. Springer, 2021, pp. 145–152.
- [79] A. Murphy *et al.*, “Representing financial data streams in digital simulations to support data flow design for a future digital twin,” *Robotics and Computer-Integrated Manufacturing*, vol. 61, p. 101853, 2020.

- [80] C. H. dos Santos *et al.*, “Decision support in productive processes through des and abs in the digital twin era: a systematic literature review,” *International Journal of Production Research*, pp. 1–20, 2021.
- [81] E. Negri *et al.*, “Field-synchronized digital twin framework for production scheduling with uncertainty,” *Journal of Intelligent Manufacturing*, vol. 32, no. 4, pp. 1207–1228, 2021.
- [82] C. H. d. Santos *et al.*, “Use of simulation in the industry 4.0 context: Creation of a digital twin to optimise decision making on non-automated process,” *Journal of Simulation*, pp. 1–14, 2020.
- [83] E. Flores-García *et al.*, “Analyzing the characteristics of digital twin and discrete event simulation in cyber physical systems,” in *IFIP International Conference on Advances in Production Management Systems*. Springer, 2020, pp. 238–244.
- [84] K. Fordyce *et al.*, “Modeling and integration of planning, scheduling, and equipment configuration in semiconductor manufacturing: Part i. review of successes and opportunities,” *International Journal of Industrial Engineering: Theory, Applications and Practice*, vol. 22, no. 5, pp. 575–600, 2015.
- [85] R. S. Edis and A. Ornek, “Simulation analysis of lot streaming in job shops with transportation queue disciplines,” *Simulation Modelling Practice and Theory*, vol. 17, no. 2, pp. 442–453, 2009.
- [86] L. Mönch, “Simulation-based benchmarking of production control schemes for complex manufacturing systems,” *Control Engineering Practice*, vol. 15, no. 11, pp. 1381–1393, 2007.
- [87] A. Xanthopoulos *et al.*, “Reinforcement learning-based and parametric production-maintenance control policies for a deteriorating manufacturing system,” *IEEE Access*, vol. 6, pp. 576–588, 2017.
- [88] R. Rocchetta *et al.*, “A reinforcement learning framework for optimal operation and maintenance of power grids,” *Applied energy*, vol. 241, pp. 291–301, 2019.
- [89] M. Chik *et al.*, “Manufacturing wip management systems with automated dispatching decision: Silterra case study,” in *2020 4th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*. IEEE, 2020, pp. 1–4.

- [90] C. Ocampo-Martinez *et al.*, “Energy efficiency in discrete-manufacturing systems: insights, trends, and control strategies,” *Journal of Manufacturing Systems*, vol. 52, pp. 131–145, 2019.
- [91] L. Wang, Z. Pan, and J. Wang, “A review of reinforcement learning based intelligent optimization for manufacturing scheduling,” *Complex System Modeling and Simulation*, vol. 1, no. 4, pp. 257–270, 2021.
- [92] J. Tang and K. Salonitis, “A deep reinforcement learning based scheduling policy for reconfigurable manufacturing systems,” *Procedia CIRP*, vol. 103, pp. 1–7, 2021.
- [93] V. Mnih *et al.*, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [94] B. Mughal *et al.*, “Optimizing packet forwarding performance in multiband relay networks via customized reinforcement learning,” *IEEE Open Journal of the Communications Society*, vol. 3, pp. 973–985, 2022.
- [95] T. Zheng *et al.*, “The applications of industry 4.0 technologies in manufacturing context: a systematic literature review,” *International Journal of Production Research*, vol. 59, no. 6, pp. 1922–1954, 2021.
- [96] Y. Lu *et al.*, “Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues,” *Robotics and Computer-Integrated Manufacturing*, vol. 61, p. 101837, 2020.
- [97] R. Gupta *et al.*, “Smart contract privacy protection using ai in cyber-physical systems: Tools, techniques and challenges,” *IEEE Access*, vol. 8, pp. 24 746–24 772, 2020.
- [98] K. Y. H. Lim, P. Zheng, and C.-H. Chen, “A state-of-the-art survey of digital twin: techniques, engineering product lifecycle management and business innovation perspectives,” *Journal of Intelligent Manufacturing*, vol. 31, no. 6, pp. 1313–1337, 2020.
- [99] I. R. Uhlmann and E. M. Frazzon, “Production rescheduling review: Opportunities for industrial integration and practical applications,” *Journal of manufacturing systems*, vol. 49, pp. 186–193, 2018.
- [100] H. Khorasgani, H. Wang, and C. Gupta, “Challenges of applying deep reinforcement learning in dynamic dispatching,” *arXiv preprint arXiv:2011.05570*, 2020.

- [101] P. Fantini, M. Pinzone, and M. Taisch, “Placing the operator at the centre of industry 4.0 design: Modelling and assessing human activities within cyber-physical systems,” *Computers & Industrial Engineering*, vol. 139.
- [102] G. Grambow, D. Hieber, and R. Oberhauser, “Intelligent task assignment in industry 4.0 production processes utilizing fuzzy sets,” in *The Tenth International Conference on Intelligent Systems and Applications*. IARIA XPS Press, 2021.
- [103] D. Gyulai, B. Kádár, and L. Monosotori, “Robust production planning and capacity control for flexible assembly lines,” *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 2312–2317, 2015.
- [104] J. P. U. Cadavid *et al.*, “Machine learning applied in production planning and control: a state-of-the-art in the era of industry 4.0,” *Journal of Intelligent Manufacturing*, pp. 1–28, 2020.
- [105] S. Fosso Wamba *et al.*, “Industry experiences of artificial intelligence (ai): benefits and challenges in operations and supply chain management,” pp. 1493–1497, 2022.
- [106] P. D. Paraschos, G. K. Koulinas, and D. E. Koulouriotis, “Reinforcement learning for combined production-maintenance and quality control of a manufacturing system with deterioration failures,” *Journal of Manufacturing Systems*, vol. 56, pp. 470–483, 2020.
- [107] A. Xanthopoulos, G. Chnitidis, and D. Koulouriotis, “Reinforcement learning-based adaptive production control of pull manufacturing systems,” *Journal of Industrial and Production Engineering*, vol. 36, no. 5, pp. 313–323, 2019.
- [108] D. Romero *et al.*, “Towards an operator 4.0 typology: a human-centric perspective on the fourth industrial revolution technologies,” in *proceedings of the international conference on computers and industrial engineering (CIE46), Tianjin, China*, 2016, pp. 29–31.
- [109] J. Heinrich and D. Silver, “Deep reinforcement learning from self-play in imperfect-information games,” *arXiv preprint arXiv:1603.01121*, 2016.
- [110] D. Silver *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.
- [111] D. Zha *et al.*, “Douzero: Mastering doudizhu with self-play deep reinforcement learning,” *arXiv preprint arXiv:2106.06135*, 2021.

- [112] P. Ecoffet *et al.*, “Policy search with rare significant events: Choosing the right partner to cooperate with,” *arXiv preprint arXiv:2103.06846*, 2021.
- [113] L. Mönch, R. Uzsoy, and J. W. Fowler, “A survey of semiconductor supply chain models part i: semiconductor supply chains, strategic network design, and supply chain simulation,” *International Journal of Production Research*, vol. 56, no. 13, pp. 4524–4545, 2018.
- [114] H. Allgeier *et al.*, “Simulation-based evaluation of lot release policies in a power semiconductor facility—a case study,” in *2020 Winter Simulation Conference (WSC)*. IEEE, 2020, pp. 1503–1514.
- [115] F. Qiao, J. Liu, and Y. Ma, “Industrial big-data-driven and cps-based adaptive production scheduling for smart manufacturing,” *International Journal of Production Research*, vol. 59, no. 23, pp. 7139–7159, 2021.
- [116] M. Panzer and B. Bender, “Deep reinforcement learning in production systems: a systematic literature review,” *International Journal of Production Research*, vol. 60, no. 13, pp. 4316–4341, 2022.
- [117] Z. Kang, C. Catal, and B. Tekinerdogan, “Machine learning applications in production lines: A systematic literature review,” *Computers & Industrial Engineering*, vol. 149, 2020.
- [118] V. H. Remko, “Research opportunities for a more resilient post-covid-19 supply chain—closing the gap between research findings and industry practice,” *International Journal of Operations & Production Management*, vol. 40, no. 4, pp. 341–355, 2020.
- [119] V. Dohale *et al.*, “Covid-19 and supply chain risk mitigation: a case study from india,” *The International Journal of Logistics Management*, 2021.
- [120] D. Ivanov, “Predicting the impacts of epidemic outbreaks on global supply chains: A simulation-based analysis on the coronavirus outbreak (covid-19/sars-cov-2) case,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 136, 2020.
- [121] C. Rumbaugh *et al.*, “Review of the effect of covid-19 on the american semiconductor industry supply chain,” in *Proceedings of the International Conference on Industrial & Mechanical Engineering and Operations Management Dhaka*, 2020, pp. 26–27.
- [122] B.-A. Han and J.-J. Yang, “Research on adaptive job shop scheduling problems based on dueling double dqn,” *IEEE Access*, vol. 8, 2020.

- [123] X. Lu *et al.*, “Data-driven optimal control of operational indices for a class of industrial processes,” *IET Control Theory & Applications*, vol. 10, no. 12, pp. 1348–1356, 2016.
- [124] C. E. Mazgualdi *et al.*, “A deep reinforcement learning (drl) decision model for heating process parameters identification in automotive glass manufacturing,” in *International Conference on Artificial Intelligence & Industrial Applications*. Springer, 2020, pp. 77–87.
- [125] M.-A. Dittrich and S. Fohlmeister, “Cooperative multi-agent system for production control using reinforcement learning,” *CIRP Annals*, vol. 69, no. 1, pp. 389–392, 2020.
- [126] M. Mowbray, D. Zhang, and E. A. D. R. Chanona, “Distributional reinforcement learning for scheduling of (bio) chemical production processes,” *arXiv preprint arXiv:2203.00636*, 2022.
- [127] Y. Zhang *et al.*, “Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems,” *Robotics and Computer-Integrated Manufacturing*, vol. 78, 2022.
- [128] D. Shi *et al.*, “Intelligent scheduling of discrete automated production line via deep reinforcement learning,” *International journal of production research*, vol. 58, no. 11, pp. 3362–3380, 2020.
- [129] C. C. Beltran-Hernandez *et al.*, “Variable compliance control for robotic peg-in-hole assembly: A deep-reinforcement-learning approach,” *Applied Sciences*, vol. 10, no. 19, 2020.
- [130] T. M. Moerland *et al.*, “A0c: Alpha zero in continuous action space,” *arXiv preprint arXiv:1805.09613*, 2018.
- [131] V. Samsonov, K. B. Hicham, and T. Meisen, “Reinforcement learning in manufacturing control: Baselines, challenges and ways forward,” *Engineering Applications of Artificial Intelligence*, vol. 112, 2022.

APPENDIX A THE SIMULATION MODEL

The building scheme of the simulation model

The simulation model is built through Arena Simulation. It adopts the concept of hybrid simulation of both Discrete-event and Agent-based simulation. In addition, it applies the data-driven scheme for controlling products' flow, resources' behaviour, and agents' working logic. Figure A.1 shows the data structure of the simulation model. The icons at each group of data shows how different groups are linked. For instance, Recipes are linked with Equipment as each recipe requires specific equipment type for processing and may be dedicated/restricted to/from specific equipment within the same equipment type station. Figure A.2 shows the proposed agent-based modeling scheme for the use case. The model consists of 4 main types of agents each has certain responsibilities. Agents behavior can be changed through the driving data in the database.

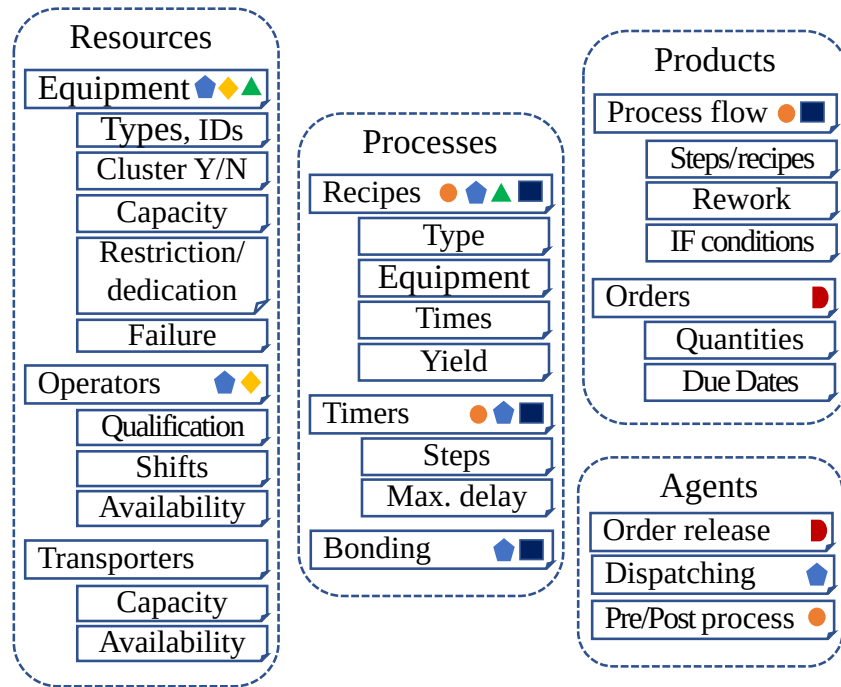


Figure A.1 The data structure of the simulation model

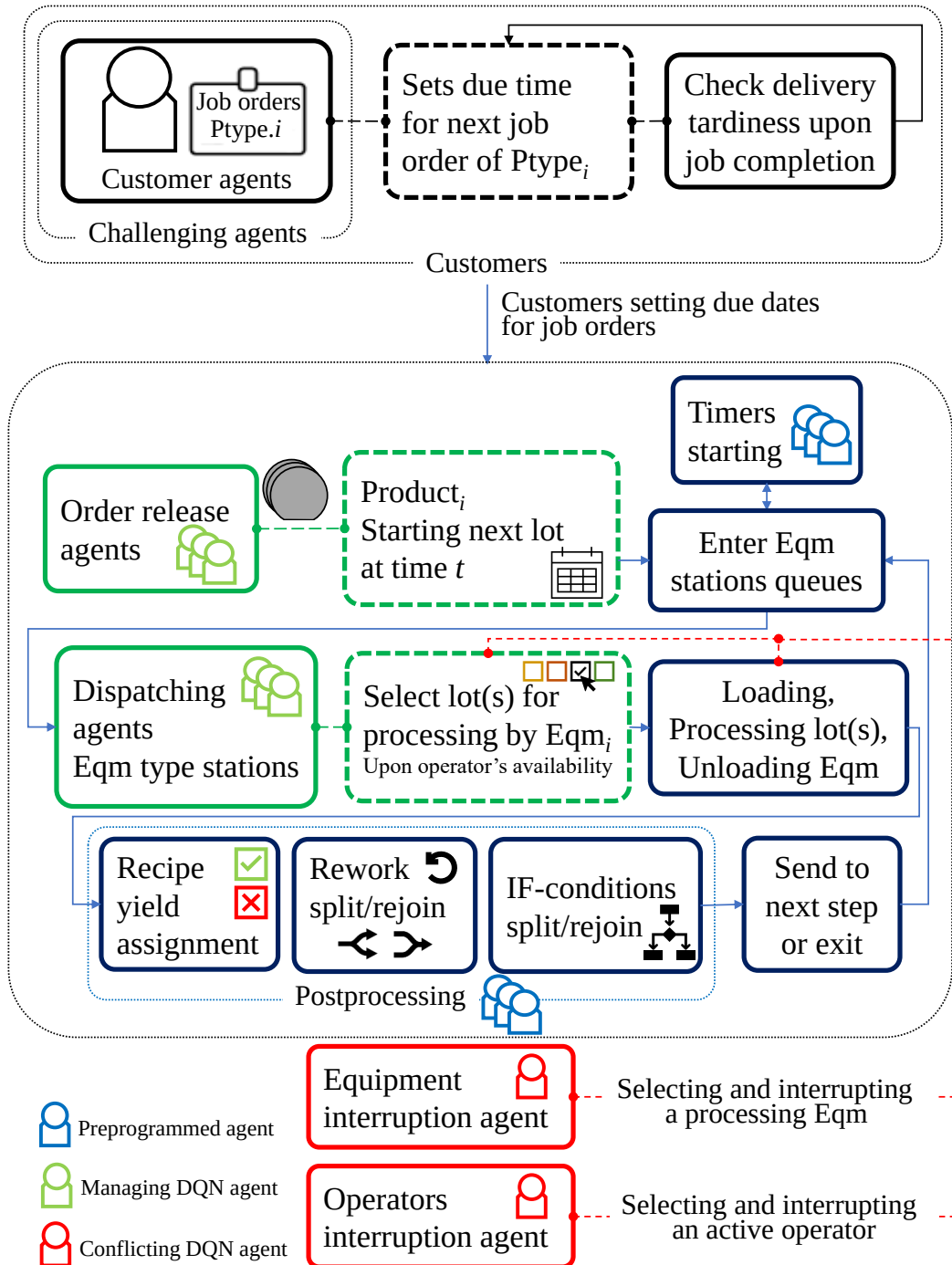


Figure A.2 The agents working logic within the simulation model

Samples from the input data of the model

The data are stored in an Excel file that is accessed through Arena at each run initialization. The data are in the form of tables each having a specific named range. Through Arena, the

module "File" is used to access the named ranges and to assign it to expressions or variables to be used through the run execution.

Sample from the process flow of one of the products

Figure A.3 shows the process flow of one of the products that are considered in the use case. The blurred parts of the figure contains classified information related to the real system. As shown in the figure, steps are identified within the model by the Step ID through which all the related information is accessed. Through the Excel file that is used to contain these data, the process flow sheet is linked with other sheets, within the same file, that contains information for recipes', Rework, IF conditions. According to their ID, the corresponding information for them are accessed by the model.

Sample from the information related to some steps with the process flow

Figure A.4 shows the timers information for steps within the process flow of each product type. Through this table, each timer is identified by an ID which defines its product type, the steps that are among this timer, and the timer's constraint or maximum delay that a lot is permitted to be within that timer without scrapping or reworking.

Figure A.5 shows the information related to rework and IF conditions. For rework, the rework ID is defined in each product's process flow. This rework ID is used to get the percentage of having this rework and the percentage of wafers from each lot that requires rework. Similarly, The IF conditions are defined. The percentage of going through the "Then" part of the IF condition's steps is defined for each IF condition through the process flow.

Customer ID		Product ID																	
Customer name		Product name																	
Step ID	Fab ID	Recipe ID/IF	Rework ID	IF condition	Rework	Rework send to_s	Rework/If rejoin at_st	Alternative recipe		1st rejoin	2nd rejoin	Next/ Yes	Else next	IF parameter	IF cond cont	Recipe name	1st Alt	2nd Alt	
1	4	2	0	0	0			0	0										
2	4	3	1	0	1	1	2	0	0										
3	4	4	0	0	0			0	0										
4	4	5	2	0	1	4	5	0	0										
5	5	6	3	0	1	5	9	0	0										
6	5	7	4	0	1	5	9	0	0										
7	5	8	5	0	1	5	9	0	0										
8	5	9	6	0	1	5	9	0	0										
9	5	10	7	0	1	5	11	0	0										
10	4	11	8	0	1	5	11	0	0										
11	4	12	0	0	0			0	0										
12	5	13	0	0	0			30	31	13	14								
13	4	14	0	0	0			0	0										
14	4	15	0	0	0			0	0										
15	0	1	0	1	0		18	0	0			16	17		IF head				
16	5	16	0	2	0			0	0			18	0 THEN		IF content				
17	5	17	0	2	0			0	0			18	0 ELSE		IF content				
18	0	2	0	1	0		20	0	0			19	20		IF head				
19	5	18	0	2	0			0	0			20	0 THEN		IF content				
20	5	19	0	0	0			0	0						After IF				
21	4	20	0	0	0			0	0										
22	0	3	0	1	0		25	0	0			23	25		IF head				
23	6	21	0	2	0			0	0			24	0 THEN		IF content				
24	6	22	0	2	0			0	0			25	0 THEN		IF content				
25	0	4	0	1	0		28	0	0			26	28		IF head				
26	6	21	0	2	0			0	0			27	0 THEN		IF content				
27	6	22	0	2	0			0	0			28	0 THEN		IF content				
28	0	5	0	1	0		31	0	0			29	31		IF head				
29	6	25	0	2	0			0	0			30	0 THEN		IF content				
30	6	22	0	2	0			0	0			31	0 THEN		IF content				
31	4	27	0	0	0			32	0	32					After IF				
32	6	28	0	0	0			0	0										

Figure A.3 The process flow of one of the use case products

Timer ID	Max timer (hr)	# of steps	Product ID	Step ID in PF			Product	Recipe name	
				From & incl.	To & incl.	# timers		From & incl.	To & incl.
1	10.00	3	1	31	33	1			
2	8.00	3	2	27	29	1			
3		0	3	0	0	0			
4		0	4	0	0	0			
5		0	5	0	0	0			
6	5.00	2	6	8	9	1			
7	18.00	3	7	31	33	2			

Figure A.4 Sample of the timers information

Rework ID	P1		P2	
	% Rework	% wafers	% Rework	% wafers
1	0.05	0.2	0.05	0.2
2	0.02	0.3	0.02	0.25
3	0.01	0.3		
4	0.02	0.25		
5	0.01	0.3		
6	0.03	0.28		
7	0.025	0.3		
8	0.02	0.3		
9				
10				

Customer ID	Product ID	IF ID	IF condition	% True
5	1	1		0.9
5	1	2		0.8
5	1	3		0.95
5	1	4		0.8
5	1	5		0.8
6	2	1		0.8
6	2	2		0.85

Figure A.5 Sample of the rework and IF conditions information

Sample from the recipe information

Figure A.6 shows a sample of the recipes that are available for processing different products. The process flow of each product defines which recipes among the available ones are set for the steps of processing. Each recipe requires a certain equipment type to be processed within. The times of each recipe are defined for first wafer of the lot, the rest of wafers, and the average time required for operator for loading and unloading. This table is linked with the equipment information and other information related to dedication for recipes as shown in the next table.

Figure A.7 shows the dedication information of some recipes. Recipes generally are processed through a certain equipment type. For a certain recipe, some equipment may be preferred for processing than the rest within the same equipment type. This is identified through the recipes' dedication information.

Sample from the equipment information

Figure A.8 shows a sample of the equipment information. If the equipment is among a cluster tool that clusters many types, a cluster ID is defined. For each equipment, the capacity is defined in form of batch type. Equipment of a batch type 0 handle only one wafer at a time and requires manual feeding by the operator. Equipment of a batch type 1 can be loaded with the whole cassette, or even more than one cassette. Equipment of a batch type 2 process part of the cassette, hence it requires splitting the lot into unit batches according to the equipment capacity. The speed factor is individual for each equipment which means its processing time for recipes is less or more than the defined on in recipes' information by this factor.

Figure A.9 shows the sets of equipment that are included within each equipment type. A recipe may be processed through any equipment of the same type unless it has dedication.

Recipe ID	Recipe name	Eqm Type	Fab-Eqm type	Site name	Fab ID	Eqm Type #	Yield	Recipe time (min)			Dicing step	Dedicatio	Scout waf	batch type
								1st wafer	Increment	Operator				
1	END		END	1								0	0	
2				1 4		13	1	1.3	1.3	3 0		1	0	0
3				1 4		13	1	1.2	1.2	3 0		1	0	0
4				1 4		7	1	140	0	7 0		1	0	1
5				1 4		30	1	6	0	3 0		1	0	1
6				1 5		159	1	0.041667	0.041667	5 0		1	0	0
7				1 5		184	1	2.5	2.5	10 0		0	0	0
8				1 5		162	0.9996	0.9	0.9	1 0		1	0	0
9				1 5		154	1	1.5	1.5	2 0		1	0	0
10				1 5		173	0.9999	0.833333	0.833333	15 0		0	0	0
11				1 4		18	1	20	0	4 0		1	1	1
12				1 4		29	1	5.65	5.65	10 0		0	0	0
13				1 5		191	1	5	5	15 0		1	0	0
14				1 4		53	1	0.416667	0.416667	7 0		1	0	0
15				1 4		30	1	15	0	5 0		1	0	1
16				1 5		185	1	0.916667	0.916667	5 0		1	0	0
17				1 5		185	1	2.916667	2.916667	5 0		1	0	0
18				1 5		185	1	2	2	5 0		1	0	0
19				1 5		165	1	12	0	2 0		1	2	1
20				1 4		3	1	0.208333	0.208333	2 0		1	0	0
21				1 6		213	0.9992	10	0	7 0		1	0	1
22				1 6		213	1	10	0	5 0		1	0	1
23				1 6		213	0.9992	10	0	5 0		1	0	1
24				1 6		213	1	10	0	5 0		1	0	1

Figure A.6 Sample of the recipes that are considered for products

Dedication by recipe		Count # of		2849
Recipe name	Recipe ID	Eqm ID	Eqm #	Possibility
	0		47	0
	0		47	0
	0		47	0
	0		47	0
	0		47	0
	0		47	0
	0		47	0
	0		47	0
	0		47	0
	12		47	Possible
	50		47	Possible
	0		47	0
	0		47	0
	0		129	0
	0		129	0
	0		129	0
	0		129	0
	0		129	0
	0		129	0
	0		72	0
	0		26	0
	0		26	0
	0		26	0
	0		26	0

Figure A.7 Sample of the dedication of certain equipment for recipes

EqmID	Eqm type	Cluster	EqmID	Fab-Eqm type	Location Zone-bloc	Eqm ID		Eqm type	Capacity	speed	Cluster	Eqm #	Max batch size
						Arena name	Eqm #						
						r_eqm_1	1	1	4	1	1	1	1
						r_eqm_2	2	2	4	1	1	1	1
						r_eqm_3	3	3	4	0	1	1	1
						r_eqm_4	4	1	4	1	1	1	1
						r_eqm_5	5	2	4	1	1	1	1
						r_eqm_6	6	4	4	1	1	1	1
						r_eqm_7	7	5	4	0	1	1	1
						r_eqm_8	8	6	4	1	1	2	1
						r_eqm_9	9	6	4	1	1	2	1
						r_eqm_10	10	6	4	1	1	2	1
						r_eqm_11	11	6	4	1	1	2	1
						r_eqm_12	12	6	4	1	1	2	1
						r_eqm_13	13	6	4	1	1	2	1
						r_eqm_14	14	6	4	1	1	2	1
						r_eqm_15	15	7	4	1	1	2	1
						r_eqm_16	16	7	4	1	1	6	1
						r_eqm_17	17	8	4	1	1	1	1
						r_eqm_18	18	8	4	1	1	1	1
						r_eqm_19	19	9	4	1	1	1	1
						r_eqm_20	20	9	4	1	1	1	1
						r_eqm_21	21	10	4	0	1	1	1
						r_eqm_22	22	11	4	1	1	4	1
						r_eqm_23	23	12	4	1	1	2	1
						r_eqm_24	24	13	4	0	1	1	1
						r_eqm_25	25	14	4	0	1	1	3.13
						r_eqm_26	26	15	4	1	1	1	1
						r_eqm_27	27	16	4	0	1	1	1
						r_eqm_28	28	17	4	1	1	1	1

Figure A.8 Sample of the equipment information

Equipment sets for each Equipment type [ordered-combined-Arena name format]															
Arena name [equipr_r_eqm_															
Helper	Eqm type #	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1:1	1	1	4	0	0	0	0	0	0	0	0	0	0	0	0
2:1	2	2	5	0	0	0	0	0	0	0	0	0	0	0	0
3:1	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0
1:2	4	6	0	0	0	0	0	0	0	0	0	0	0	0	0
2:2	5	7	0	0	0	0	0	0	0	0	0	0	0	0	0
4:1	6	8	9	10	11	12	13	14	0	0	0	0	0	0	0
5:1	7	15	16	96	157	0	0	0	0	0	0	0	0	0	0
6:1	8	17	18	159	160	0	0	0	0	0	0	0	0	0	0
6:2	9	19	20	0	0	0	0	0	0	0	0	0	0	0	0
6:3	10	21	0	0	0	0	0	0	0	0	0	0	0	0	0
6:4	11	22	61	0	0	0	0	0	0	0	0	0	0	0	0
6:5	12	23	43	0	0	0	0	0	0	0	0	0	0	0	0
6:6	13	24	0	0	0	0	0	0	0	0	0	0	0	0	0
6:7	14	25	0	0	0	0	0	0	0	0	0	0	0	0	0
7:1	15	26	0	0	0	0	0	0	0	0	0	0	0	0	0
7:2	16	27	0	0	0	0	0	0	0	0	0	0	0	0	0
8:1	17	28	0	0	0	0	0	0	0	0	0	0	0	0	0
8:2	18	29	0	0	0	0	0	0	0	0	0	0	0	0	0
9:1	19	30	0	0	0	0	0	0	0	0	0	0	0	0	0
9:2	20	31	0	0	0	0	0	0	0	0	0	0	0	0	0
10:1	21	32	0	0	0	0	0	0	0	0	0	0	0	0	0
11:1	22	33	34	35	36	37	79	81	103	115	162	0	0	0	0
12:1	23	38	0	0	0	0	0	0	0	0	0	0	0	0	0
13:1	24	39	0	0	0	0	0	0	0	0	0	0	0	0	0
14:1	25	40	0	0	0	0	0	0	0	0	0	0	0	0	0
15:1	26	41	0	0	0	0	0	0	0	0	0	0	0	0	0
16:1	27	42	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure A.9 Sample of the equipment included within each equipment type

Sample from the operators information

Figure A.10 shows a sample of the operators information. Each operator works in a specific fabrication area and having a specific working shift as shown in the figure. Figure A.11 shows the specific working hours and breaks for each operator according to his/her working shift through the day. Figure A.12 shows the different equipment types that can be operated by each operator according to his/her qualifications or level of training.

Operator	Name	Arena name	Fab name	work shift
1		r_op_1	4	Night
2		r_op_2	4	Evening
3		r_op_3	4	Morning
4		r_op_4	4	Morning
5		r_op_5	4	Evening
6		r_op_6	4	Morning
7		r_op_7	4	Morning
8		r_op_8	4	Evening
9		r_op_9	4	Night
10		r_op_10	4	Morning
11		r_op_11	4	Night
12		r_op_12	4	Night
13		r_op_13	4	Morning
14		r_op_14	5	Night
15		r_op_15	5	Night
16		r_op_16	4	Evening
17		r_op_17	4	Evening
18		r_op_18	4	Evening
19		r_op_19	4	Night
20		r_op_20	5	Morning
21		r_op_21	5	Evening
22		r_op_22	5	Morning
23		r_op_23	4	Morning

Figure A.10 Sample of the Operators information

Operator ID	Operator name	Arena name	0.00	4.00	4.50	8.00	12.00	12.50	16.00	20.00	20.50	From Hr
			4.00	0.50	3.50	4.00	0.50	3.50	4.00	0.50	3.50	Duration (hr
1		r_op_1	1	0	1	0	0	0	0	0	0	
2		r_op_2	0	0	0	0	0	0	1	0	1	
3		r_op_3	0	0	0	1	0	1	0	0	0	
4		r_op_4	0	0	0	1	0	1	0	0	0	
5		r_op_5	0	0	0	0	0	0	1	0	1	
6		r_op_6	0	0	0	1	0	1	0	0	0	
7		r_op_7	0	0	0	1	0	1	0	0	0	
8		r_op_8	0	0	0	0	0	0	1	0	1	
9		r_op_9	1	0	1	0	0	0	0	0	0	
10		r_op_10	0	0	0	1	0	1	0	0	0	
11		r_op_11	1	0	1	0	0	0	0	0	0	
12		r_op_12	1	0	1	0	0	0	0	0	0	
13		r_op_13	0	0	0	1	0	1	0	0	0	
14		r_op_14	1	0	1	0	0	0	0	0	0	
15		r_op_15	1	0	1	0	0	0	0	0	0	
16		r_op_16	0	0	0	0	0	0	1	0	1	
17		r_op_17	0	0	0	0	0	0	1	0	1	
18		r_op_18	0	0	0	0	0	0	1	0	1	
19		r_op_19	1	0	1	0	0	0	0	0	0	
20		r_op_20	0	0	0	1	0	1	0	0	0	

Figure A.11 Sample of the Operators information for shifts

Main module groups of the simulation model

Customer agents for setting due dates

Figure A.13 shows the working logic of customer RL agents for each product type. In the use case, two different product types are considered. In case of no RL utilization for customers, the due dates are set through the input data for each product type.

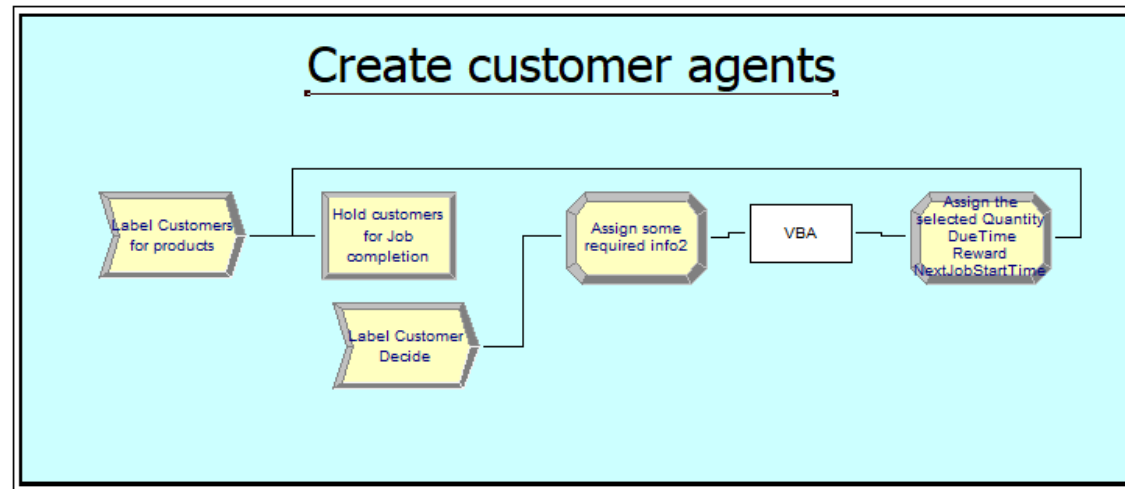


Figure A.13 Customer agents working logic

Dispatching agents creation

Figure A.14 shows the creation of dispatching agents for equipment stations and for cluster tools as well. Within a cluster tool, there are several equipment types each of having its own dispatching agent. The dispatching agent of the cluster tool dispatches waiting lots that were dispatched by each of its equipment dispatching agents. Dispatching agents apply heuristics or RL according to the setting of the simulation for dispatching.

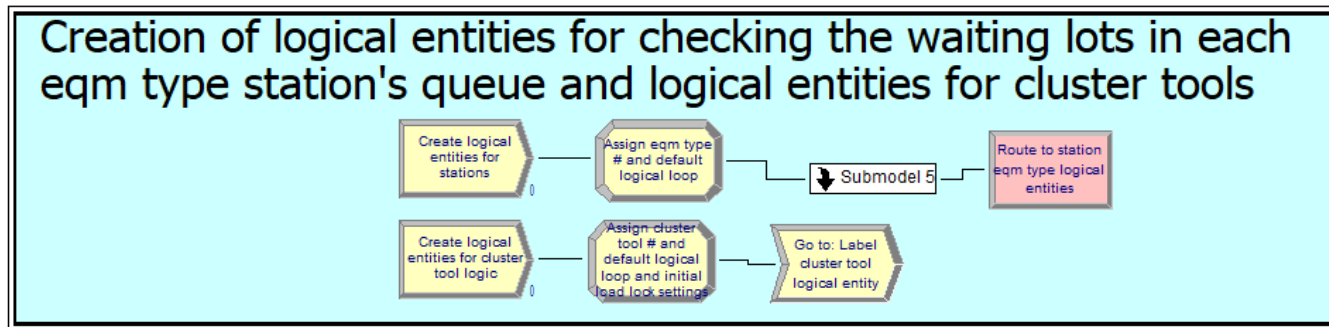


Figure A.14 Dispatching agents creation

Resources interruption agents creation and working logic

Figure A.15 shows the creation of interruption agents for both operators and equipment. These agents select resources to interrupt either randomly or based on RL according to the simulation setting.

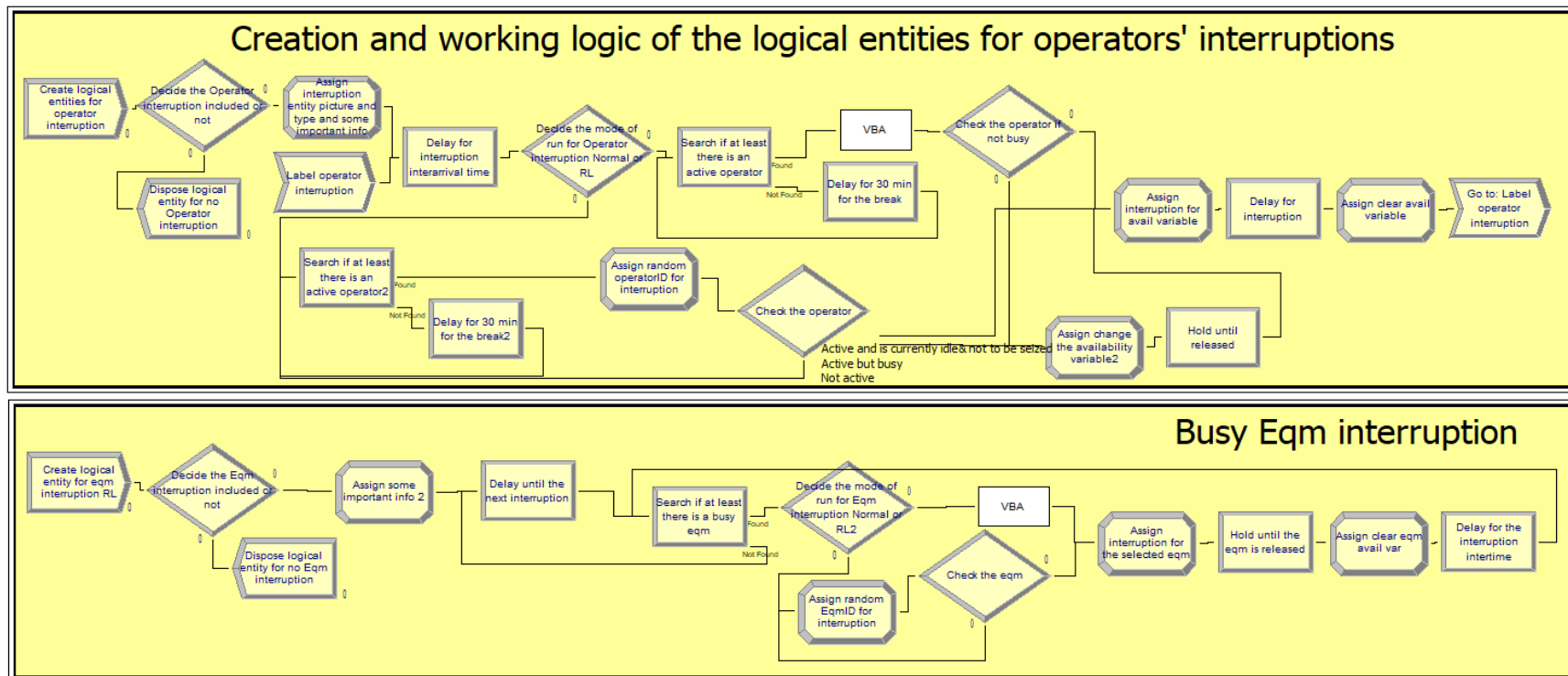


Figure A.15 Interruption agents creation and working logic

Equipment uptime/downtime agents creation and working logic

Figure A.16 shows the creation and working logic of equipment uptime/downtime agents. These agents handle assigning equipment failure and handling different events such as routing any products that are being processed while equipment is to be failed. These agents are preprogrammed only according to time distributions for each equipment type for both uptime and downtime.

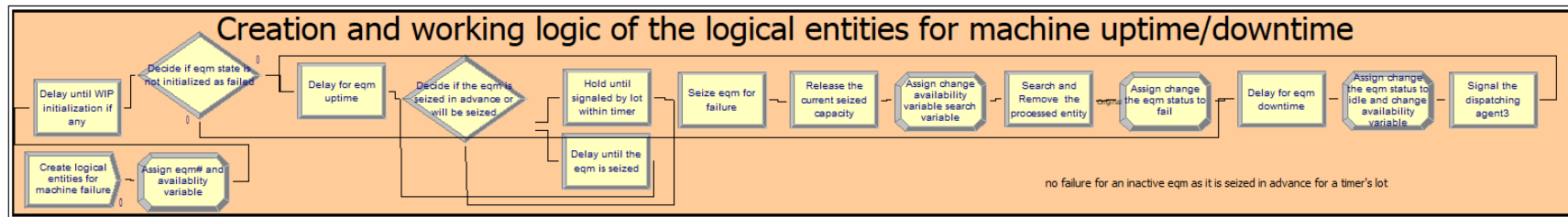


Figure A.16 Equipment uptime/downtime agents creation and working logic

Moving averages calculation of for agents' reward assignment

Figure A.17 shows the calculation moving averages of some reward components for the different RL agents. The created entities do the calculation according to the set window size and time step for each moving average component.

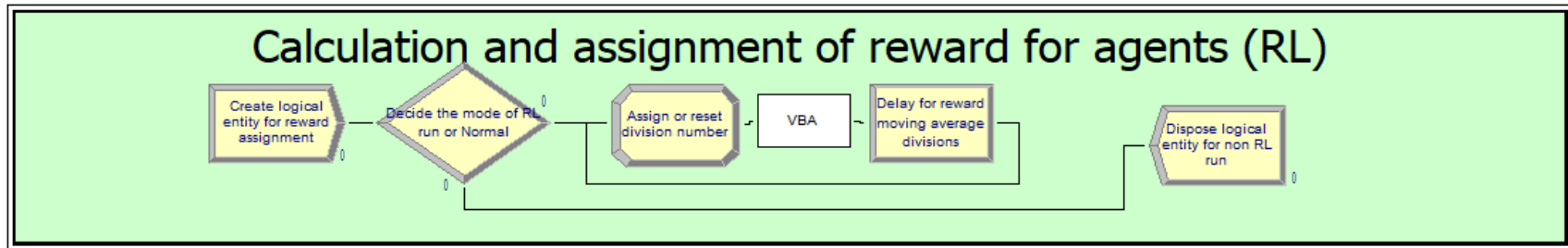


Figure A.17 Moving averages calculation of for agents' reward

SR calculation for each product type

Figure A.18 shows the calculation for SR for each product type. Through this group of modules, the job orders information is updated when job are fulfilled.

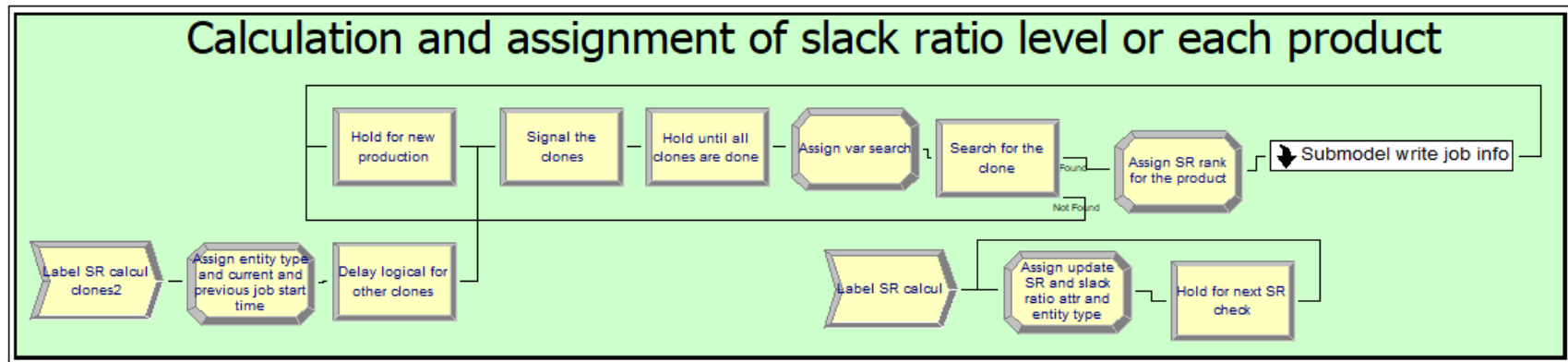


Figure A.18 SR calculation for each product type

Order release for new products

Figure A.19 shows the group of modules that are responsible for releasing or starting new lots for each product. The mode for order release is based on interarrival times that are driven by time distributions or by the order release agent for each product type. Upon creation of a new lot, some information related to rework and scrap calculation and release times are updated/saved. The created lot is assigned with important attributes for identifications. Then, the lot is sent to its first station for processing according to its process flow. A transporter resource is utilized to enter/exit to/from different fabrication areas. In this model, there are 3 different fab areas.

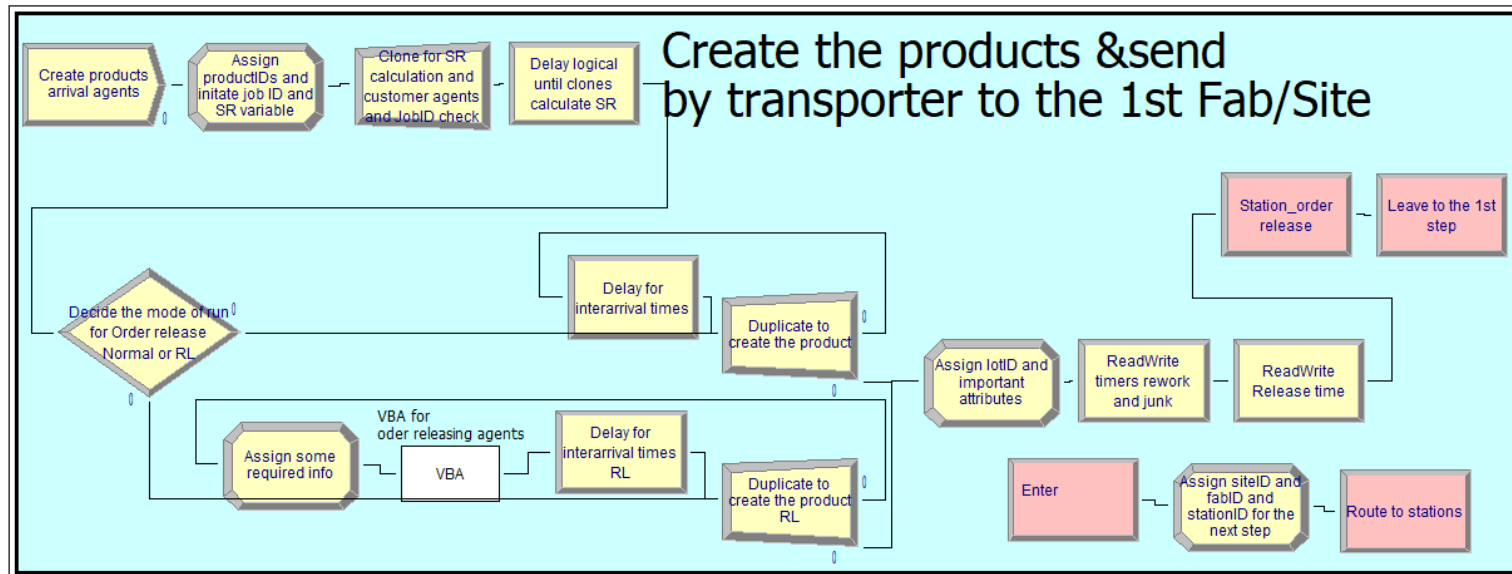


Figure A.19 Order release of products

Preparing lots to enter equipment stations

Figures A.20 and A.21 show the group of modules that are responsible for preparing and sending lots to enter the equipment stations. This include certain situations that need special handling such as scout wafers and bonding steps. For lots that need a scout wafer, a single wafer has to be sent through some processing steps before the rest of the lot to avoid scrapping. For bonding steps, two lots have to be sent together through the same step of bonding. Moreover, these bonding steps may be within a timer. After these situations are handled, the important information is assigned for lots before sent for waiting within equipment stations' queues.

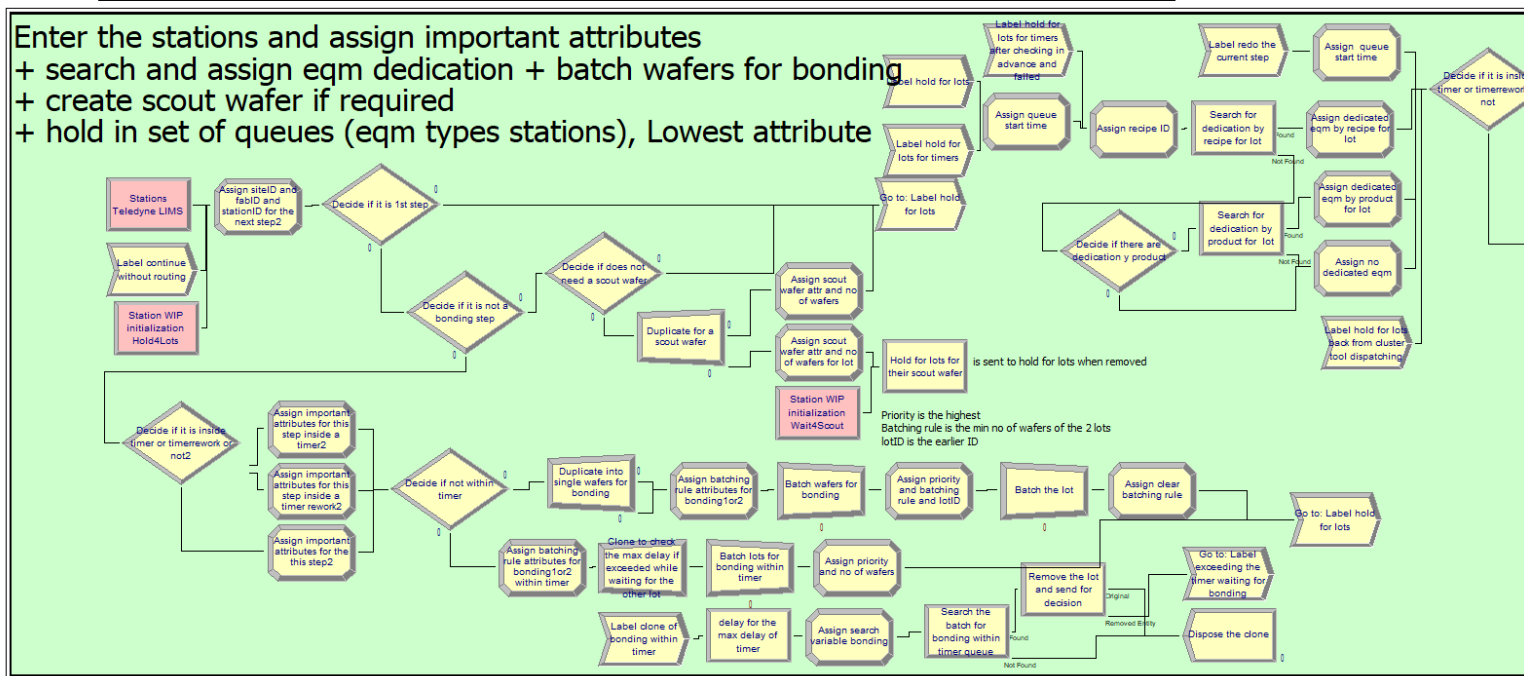


Figure A.20 Preparing lots to enter equipment stations

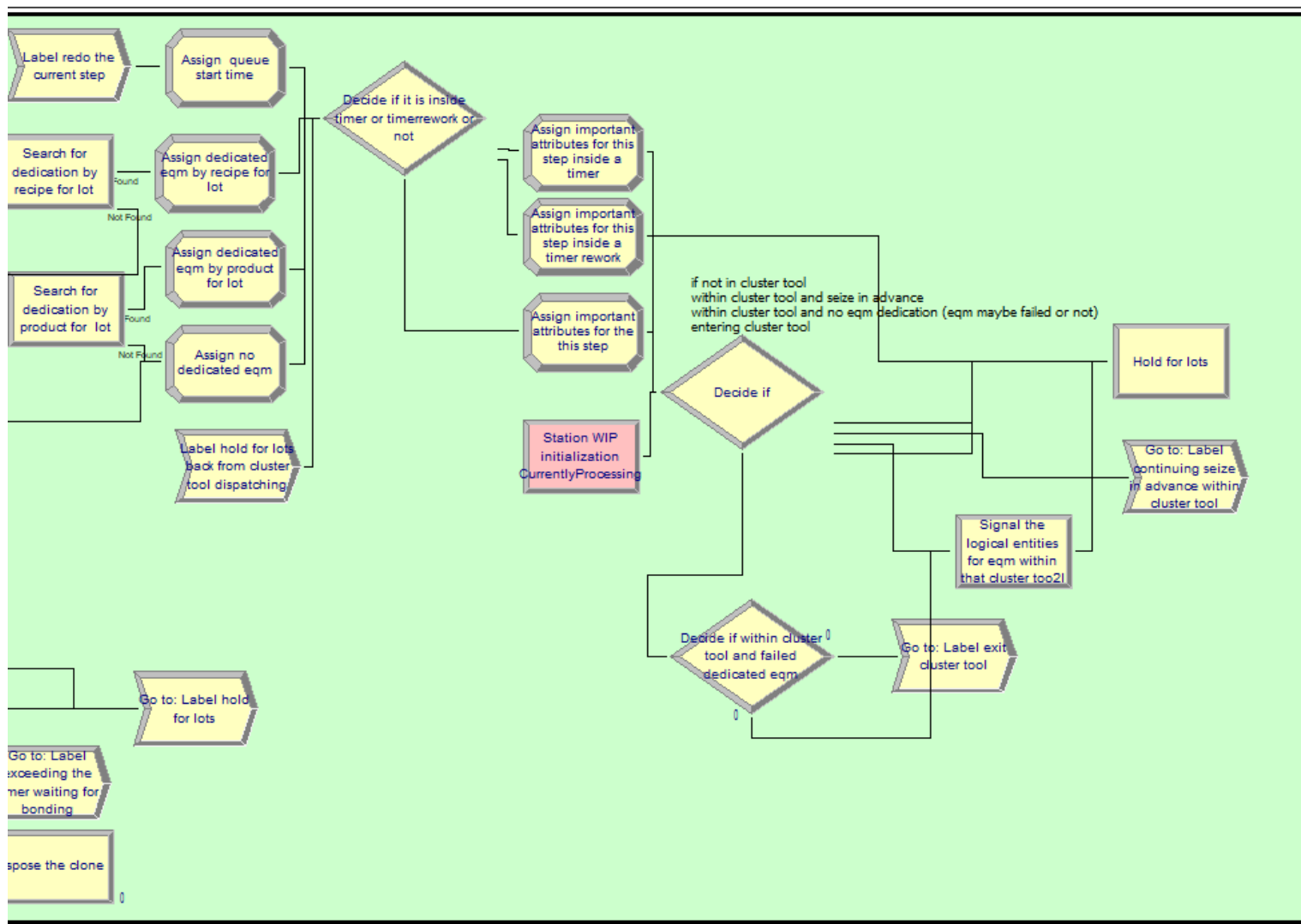
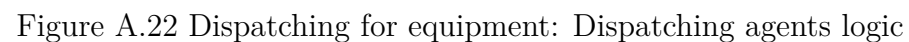


Figure A.21 Sending lots to enter equipment stations

Figures A.22 and A.23 show the group of modules in which dispatching agents pick the lot to be dispatched. This is done according to the availability of the equipment, considering recipe dedication/restrictions, and the qualified operator for loading the equipment. The dispatching agent applies a preprogrammed logic through dispatching rules, or applies RL for selecting the lot(s) to be dispatched.



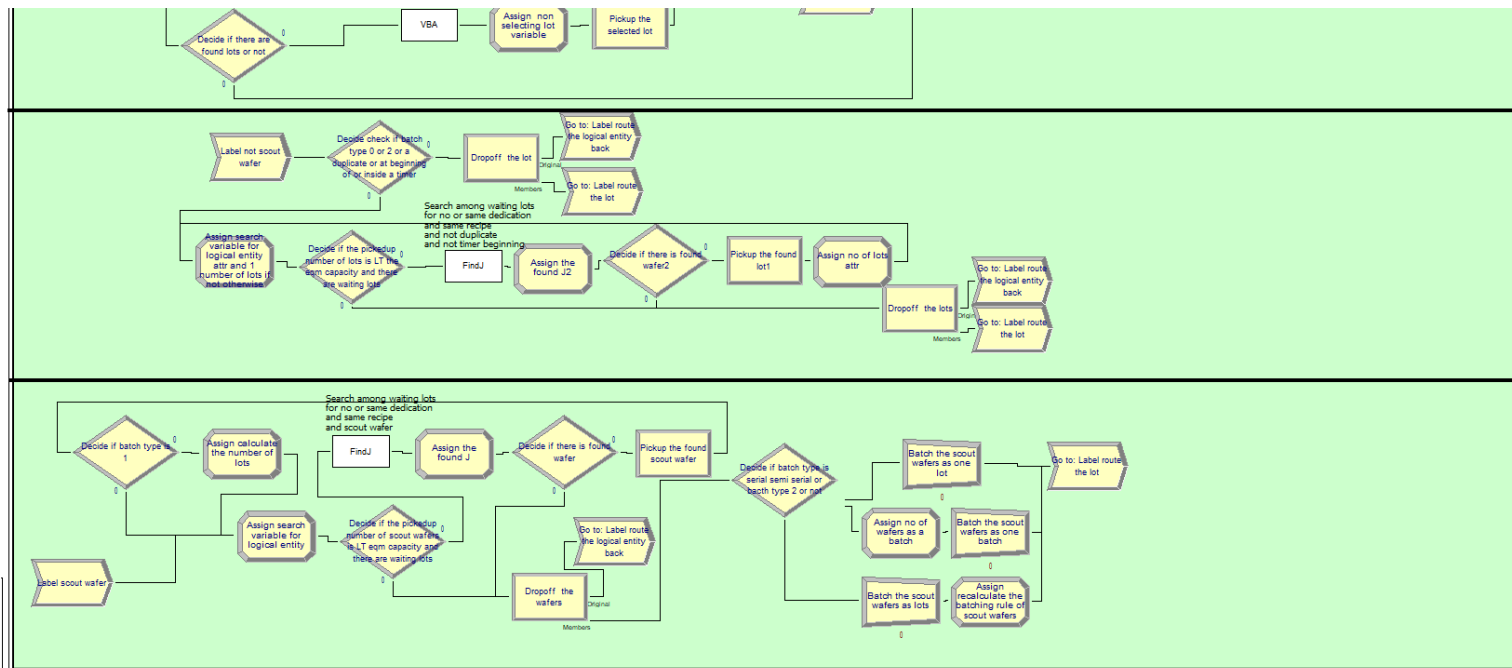


Figure A.23 Dispatching for equipment: Dispatched lots

Dispatching for cluster tools: included equipment types

Figure A.24 shows the group of modules in which dispatching agents for equipment types within clusters select lots from their corresponding queues to be available for dispatching by the main cluster tool agent within the cluster. This working logic includes handling specific situations such as scout wafers and timers that need seize or hold in advance.

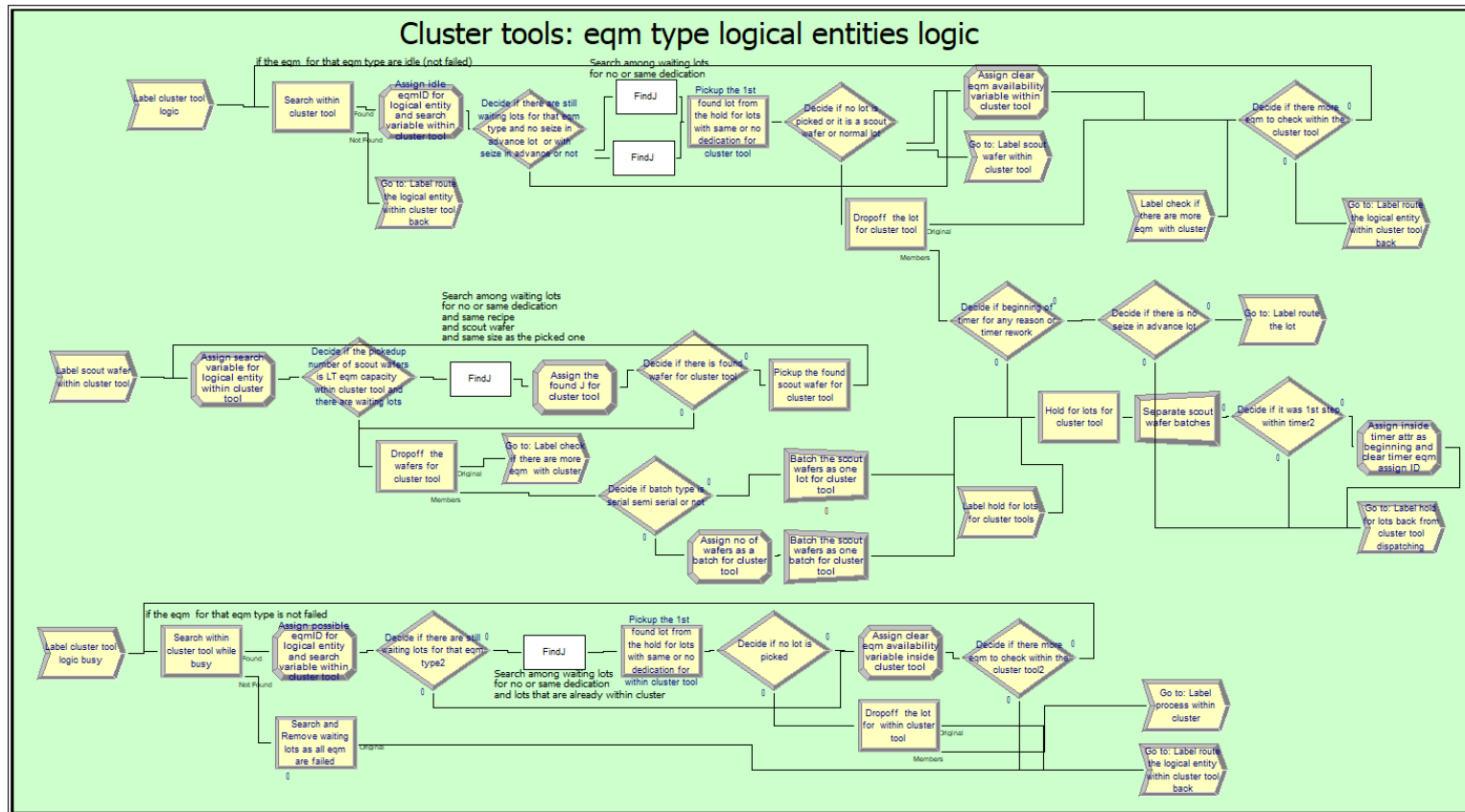


Figure A.24 Dispatching for equipment within clusters: included equipment types

Figures A.25 and A.26 show the group of modules in which dispatching agents of the main cluster select lots from which the included equipment types agents selected for dispatching. These agents apply either preprogrammed logic such as heuristic dispatching rules or RL for dispatching according to the setting of the simulation.



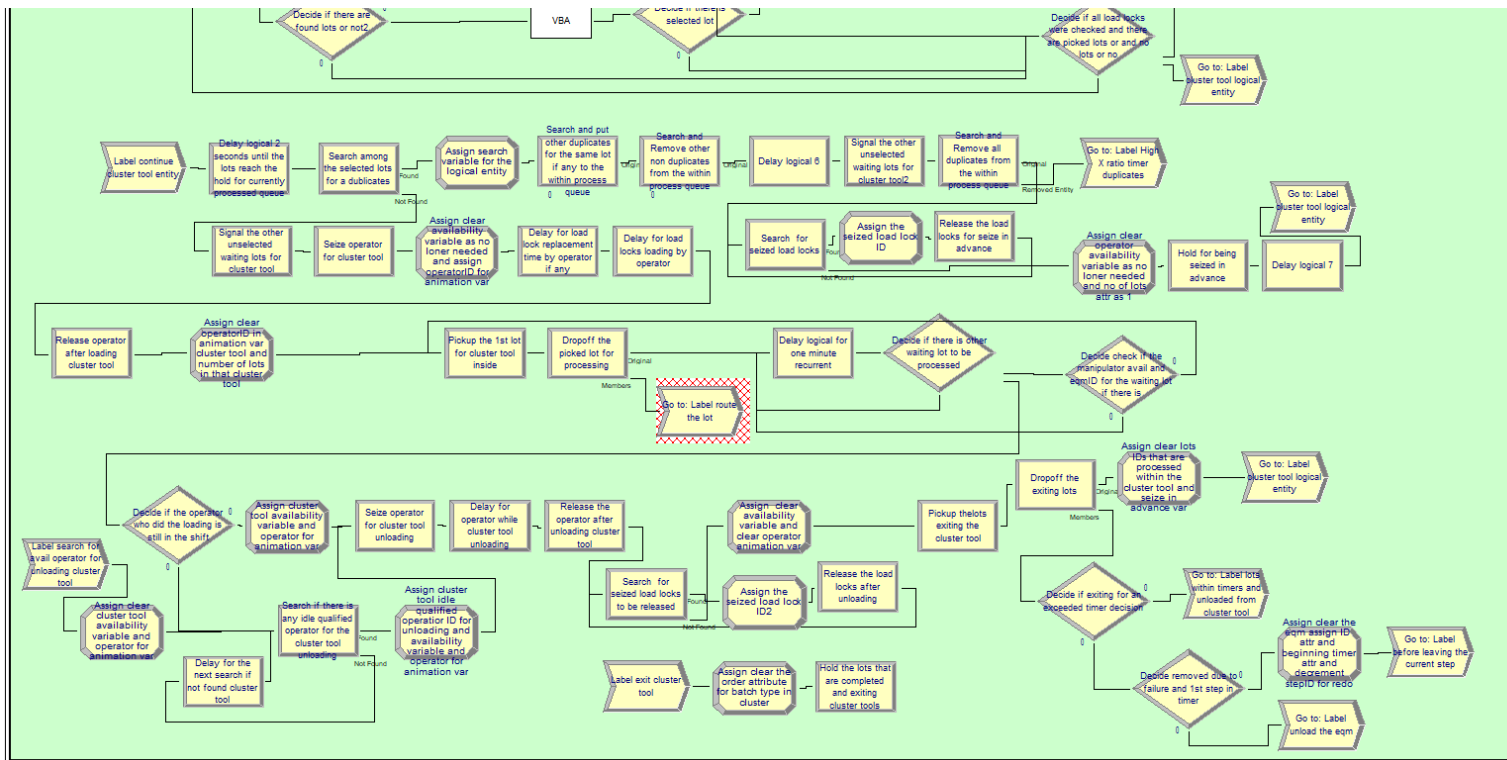


Figure A.26 Dispatching for equipment within clusters: routing dispatched lots

Seizing operator and equipment and delaying for processing time

Figures A.27 and A.28 show the group of modules in which the dispatched lots are sent to seize operators and equipment. According to the loading scheme of the equipment, the operators are kept seized or released. Then the lots are held according to the processing time of their recipe.

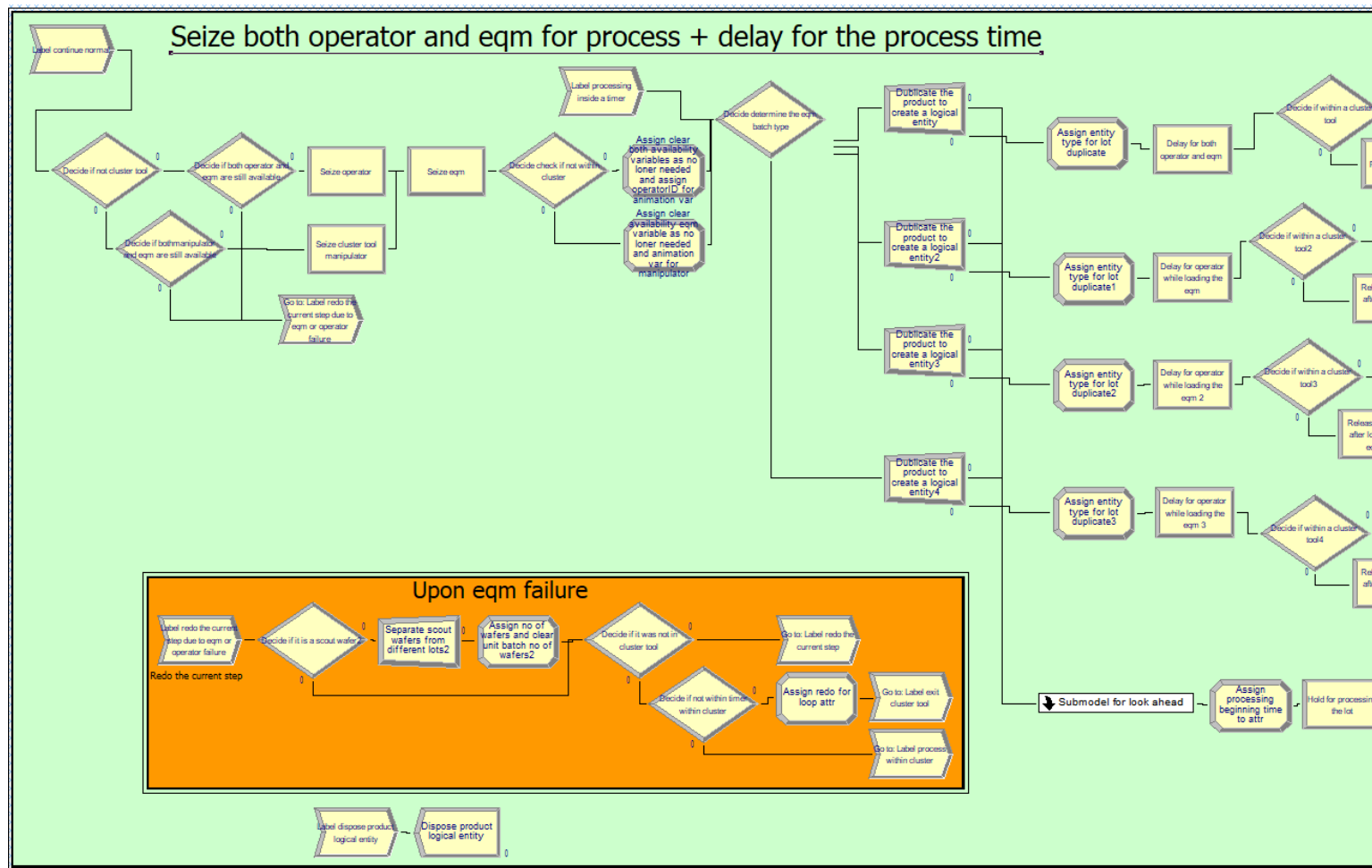


Figure A.27 Seizing resources

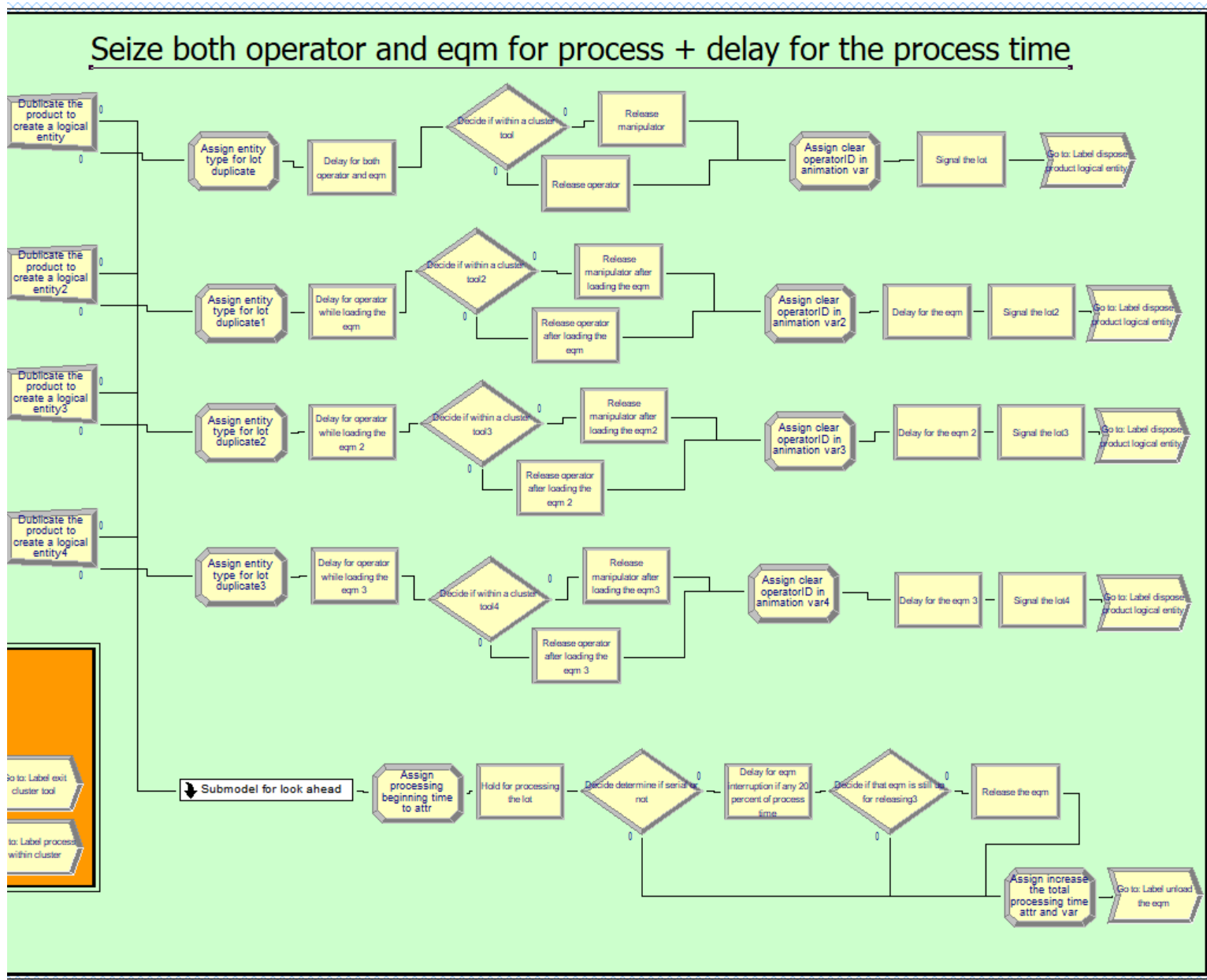


Figure A.28 Delaying for processing

Unloading the equipment and releasing operator

Figure A.29 shows the group of modules in which the processed lots are unloaded from the equipment. This is done by the operator who did the loading or any qualified operator if he/she is not in the working shift.

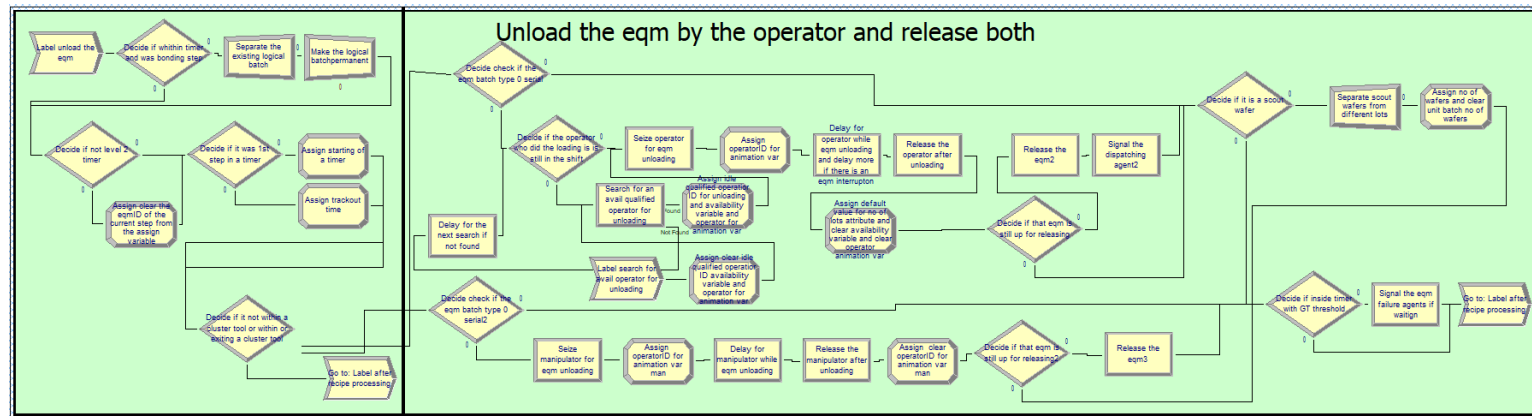


Figure A.29 Unloading equipment

Yield assignment for processed lots

Figure A.30 shows the group of modules in which wafers from the processed lots are scrapped according to the yield percentage of their recipe. These group of modules consider special situations such as if all wafers from a rework part of a lot are scrapped. In that case, the original part should be signaled to continue its process flow in case it does not have any other rework parts.

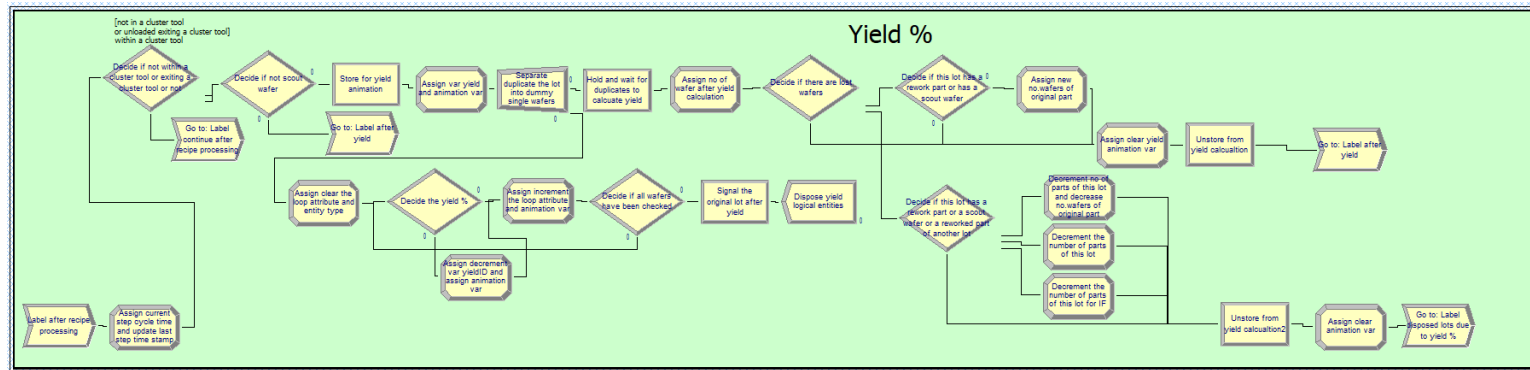


Figure A.30 Recipe yield assignment

Rework/rejoin

Figure A.31 shows the group of modules in which wafers that need to be reworked are split from their parent lot and sent for reworking. For wafers that finished their rework steps, they are held to rejoin their parent lot so that they continue their process flow. The same happens in case of scout wafer as the rest of the lot rejoins its scout wafer after processing through the steps that were checked by their scout wafer.

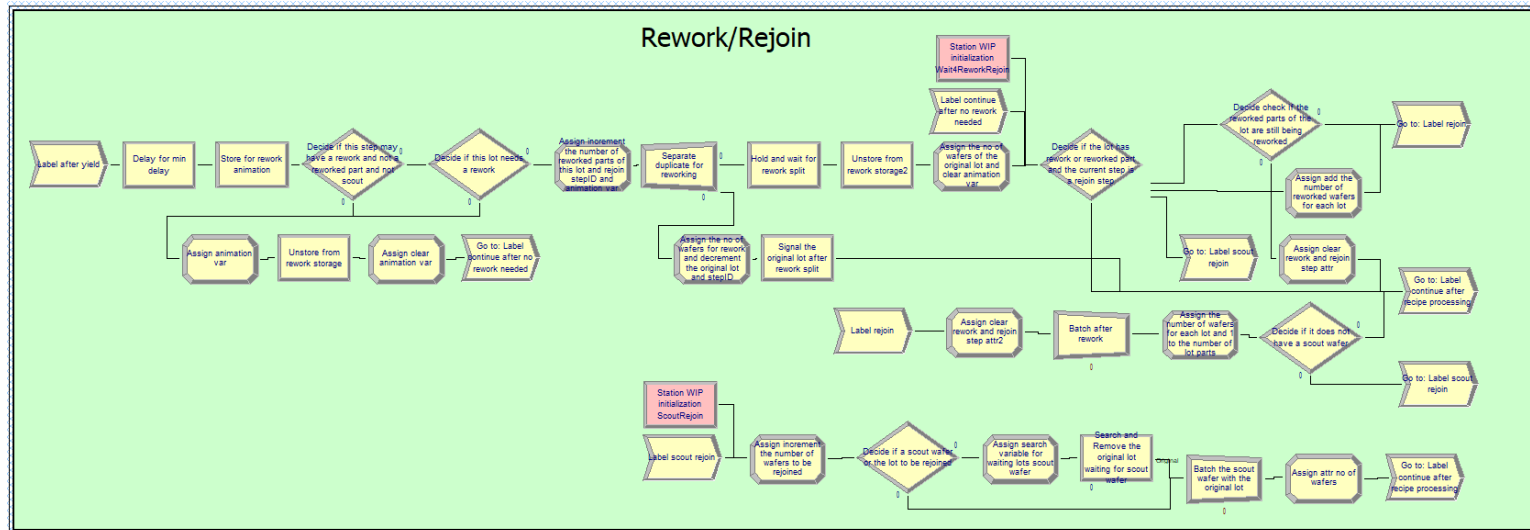


Figure A.31 Rework/Rejoin

IF Conditions and assignment of next step

Figure A.32 shows the group of modules in which lots that are entering an IF Conditions are checked for splitting. According to the percentage of each branch of the IF condition, lots are split and each split is sent to its corresponding processing step. When the IF condition steps are done for both splits, lots' wafers are rejoined and continue through their process flow.

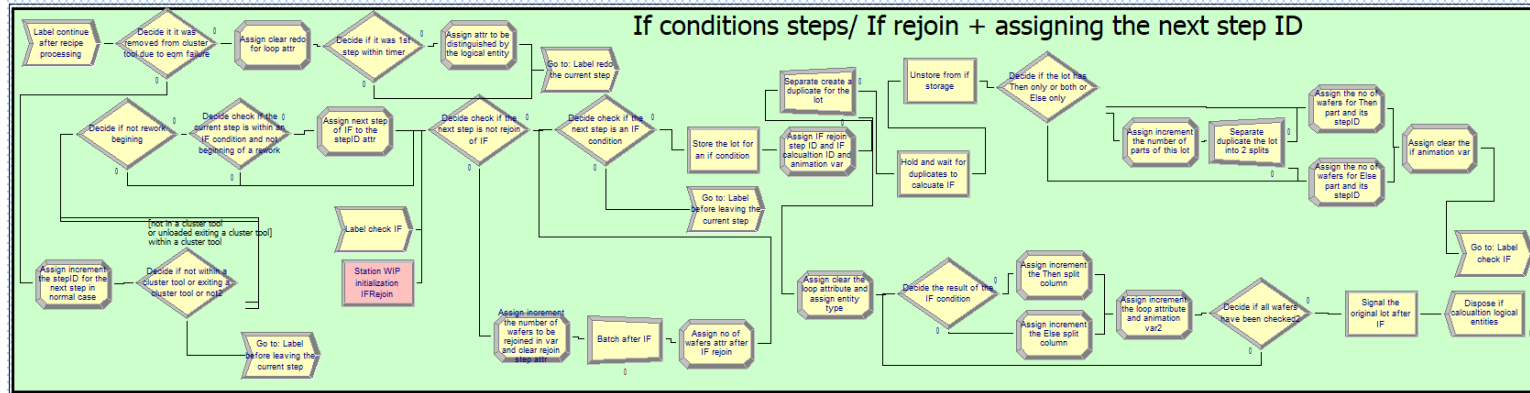


Figure A.32 IF Conditions and assignment of the next step

Leaving the current step and transferring to the next one

Figure A.34 shows the group of modules in which lots are transferred to their next step. The transfer differs if it is within the same fabrication area or between different areas. The difference is in the transporter for each transfer.

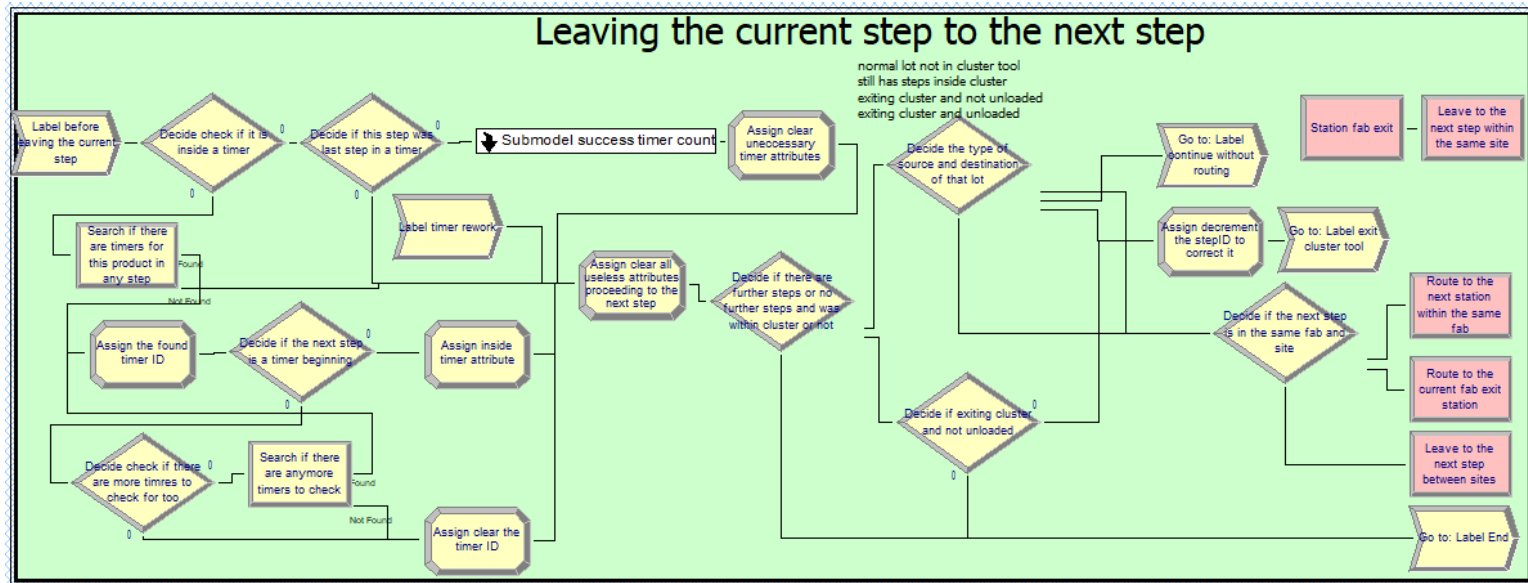


Figure A.33 Transferring lots to their next step

Leaving the current step and transferring to the next one

Figure A.34 shows the group of modules in which lots are transferred to their next step. The transfer differs if it is within the same fabrication area or between different areas. The difference is in the transporter for each transfer.

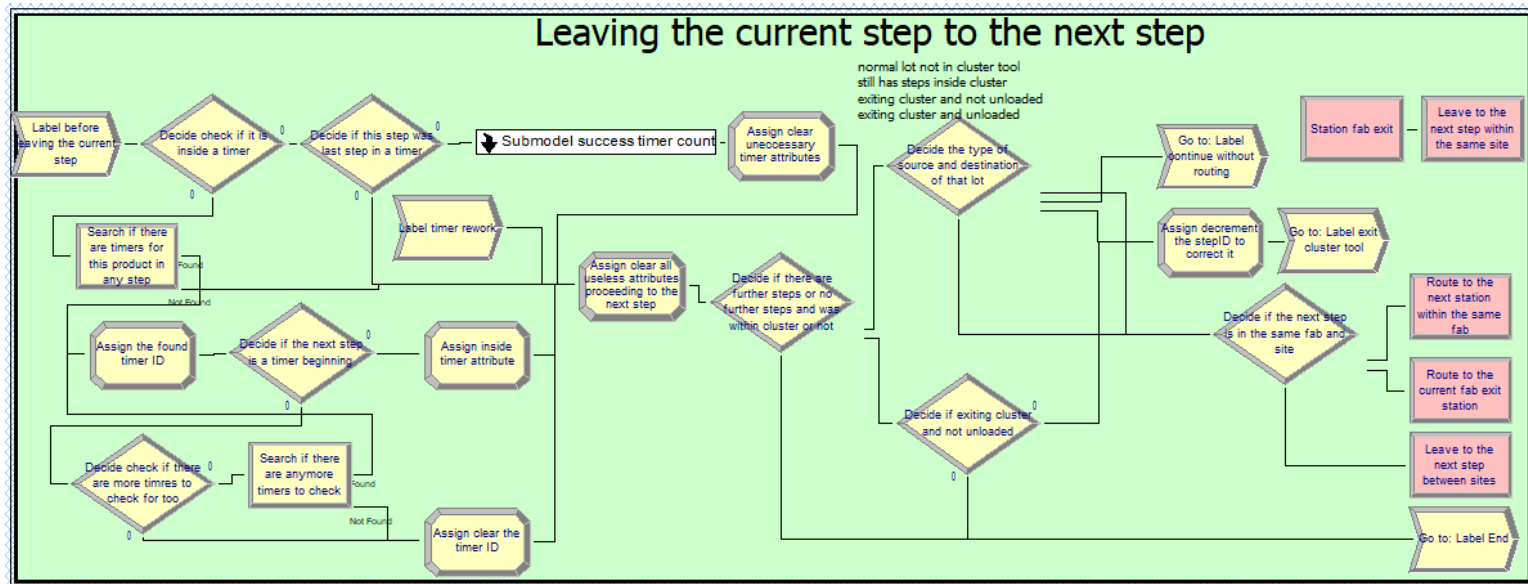


Figure A.34 Transferring lots to their next step

Check for Timers steps

Figure A.35 shows the group of modules in which lots that are dispatched for processing are checked if they will start a timer, within a timer, or no timer. In case the lot starts a timer or within a timer, they are routed accordingly. The routing differs if the mode of dispatching is RL as some timers aspects are suppressed, specifically the seize or hold in advance.

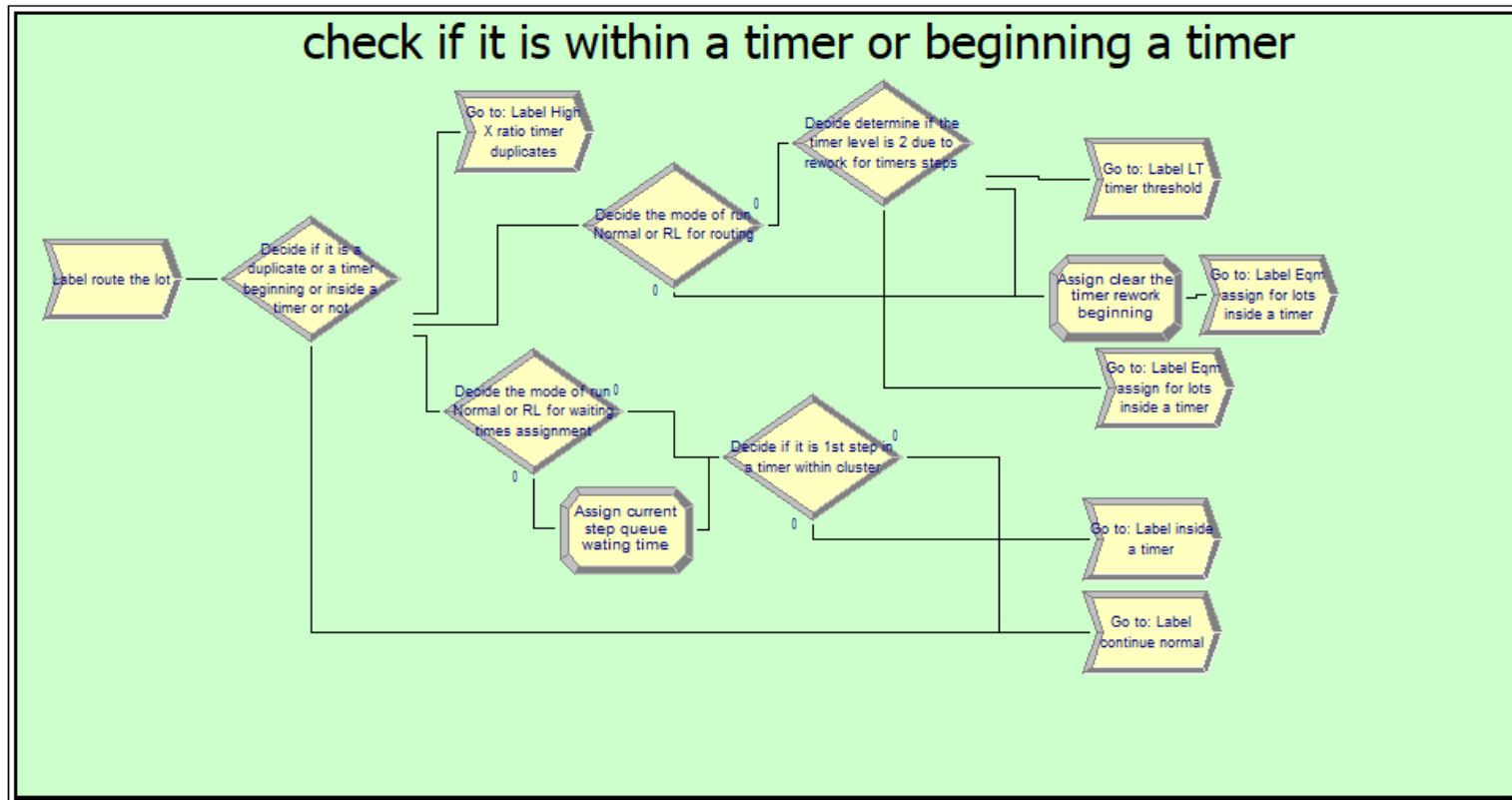


Figure A.35 Check for timers steps

Define timers' equipment and processing criticality

Figures A.36 and A.37 show the group of modules in which lots that will start a timer are check to define their equipment through timer's processing and the timers' criticality.

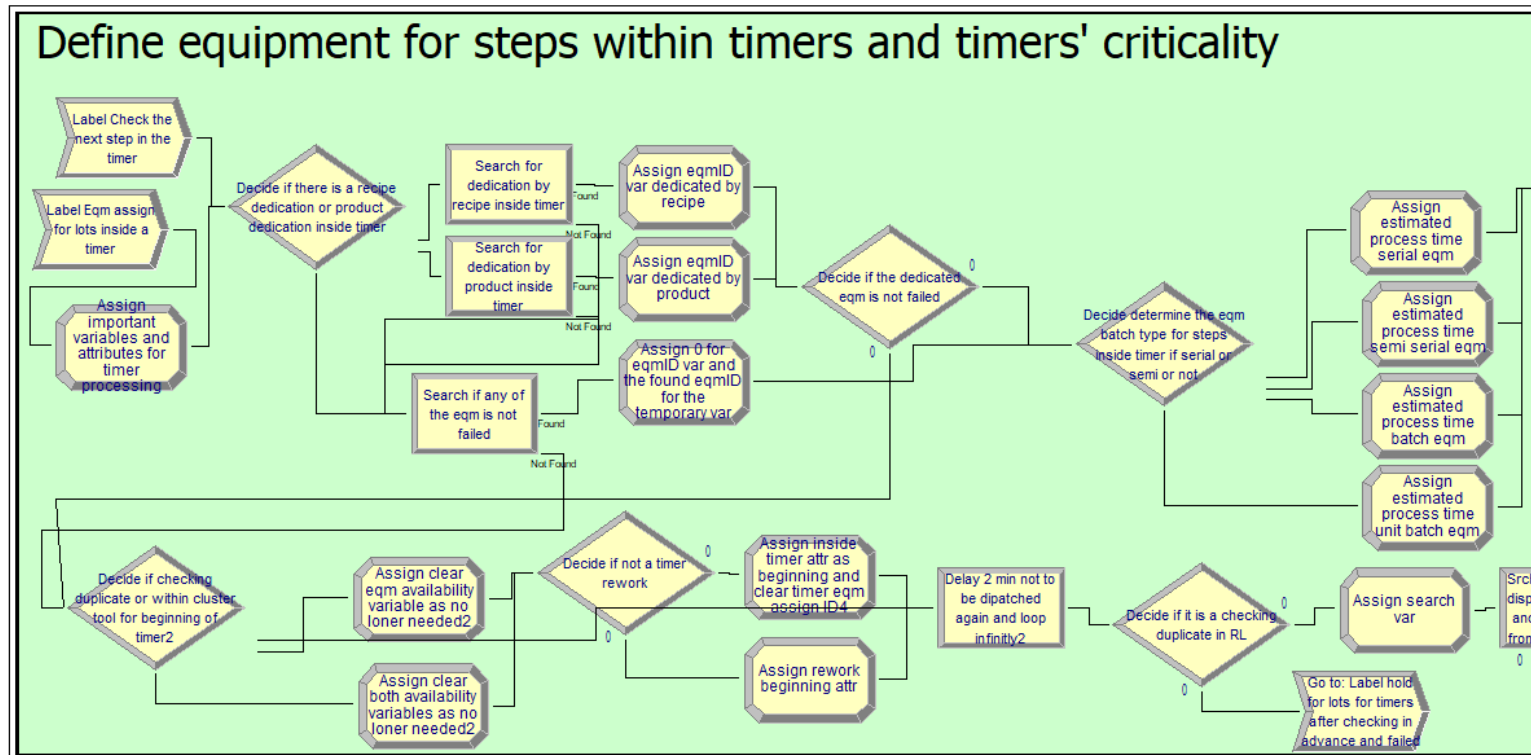


Figure A.36 Define equipment for timers steps

Define equipment for steps within timers and timers' criticality

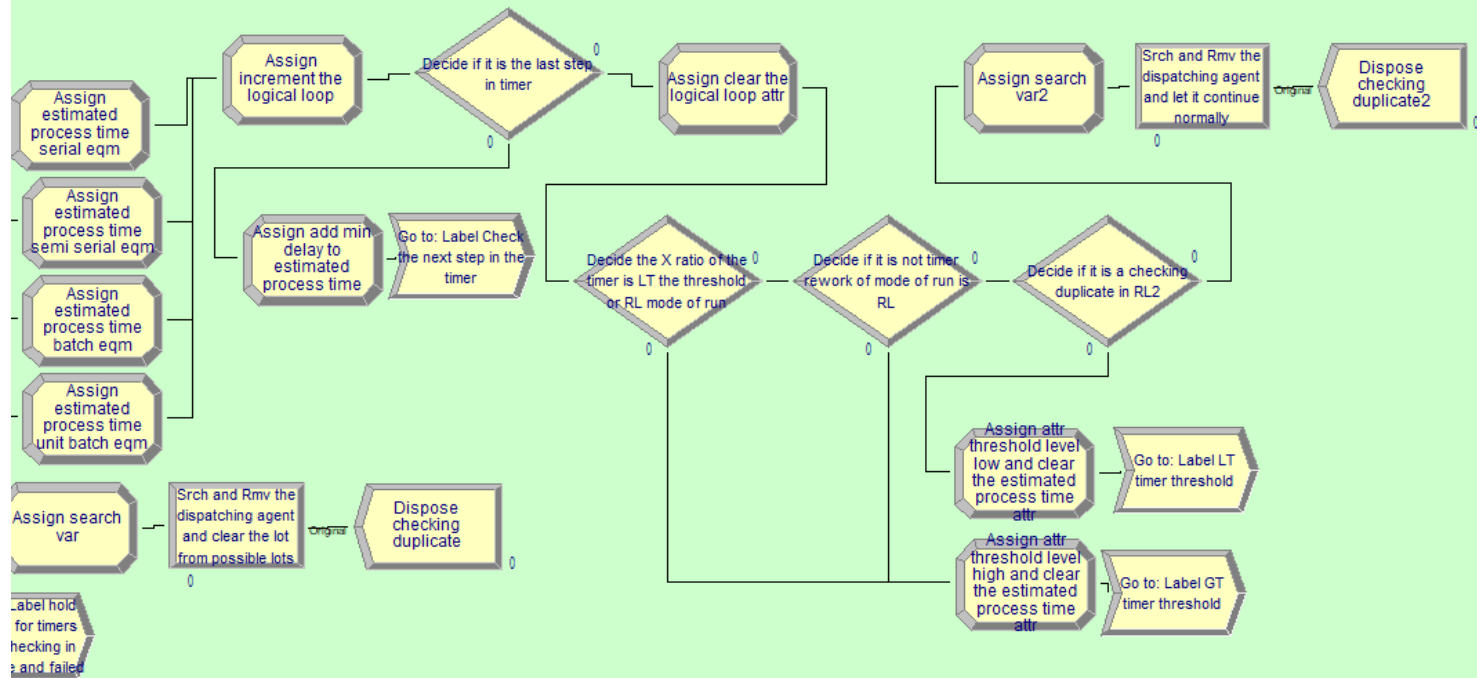


Figure A.37 Check for timers' criticality

High/low timer's criticality

Figures A.38 and A.39 show the group of modules in which lots go through either of them according to their calculated timer's criticality. In case of high criticality timer, the equipment through timer's steps are seized in advance first before processing the timer's first step. In case of low criticality, only the equipment are checked whether they are not failed before starting the timer. Accordingly, the lots are routed through the model. In case of RL setting, the seize in advance is suppressed.

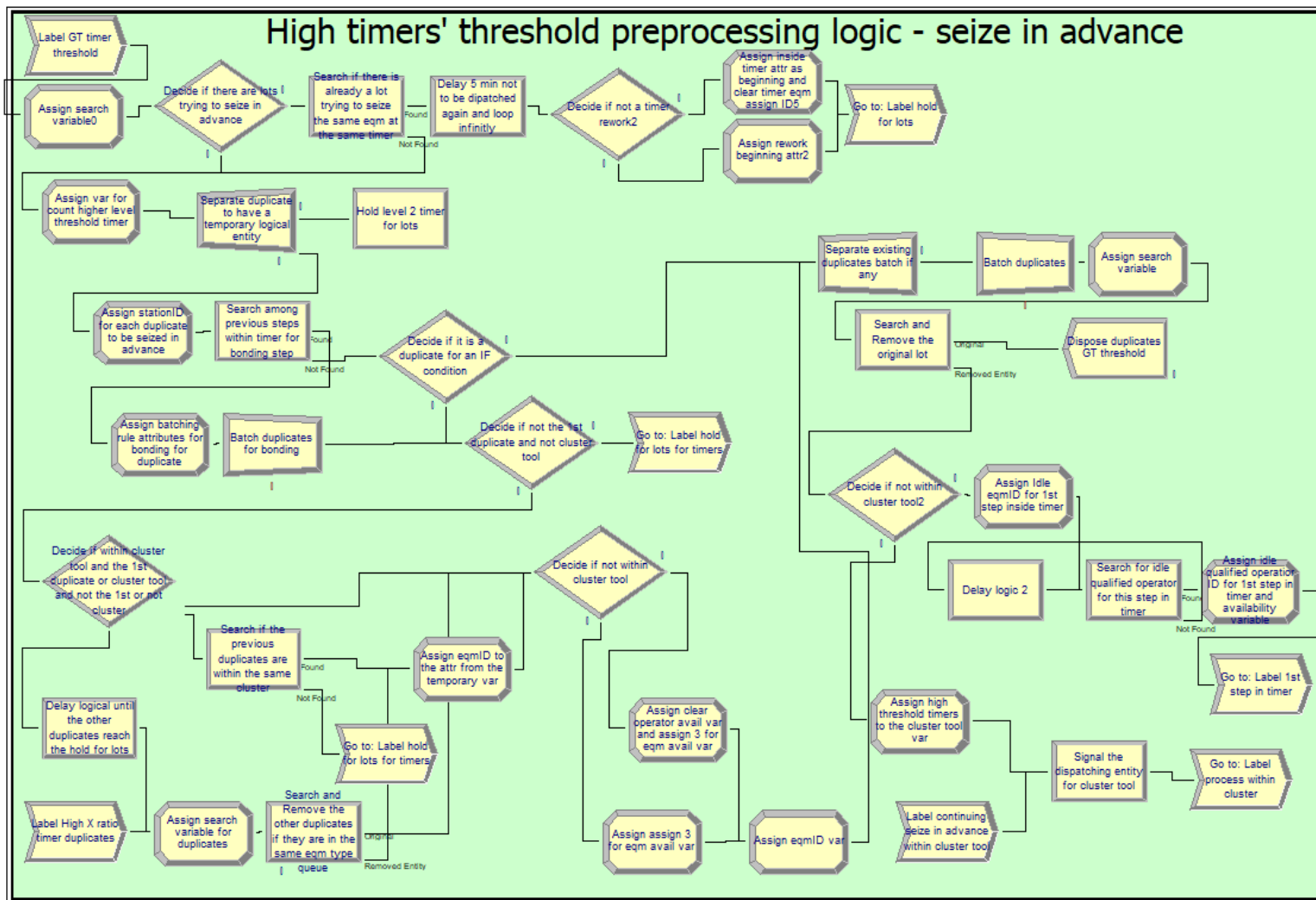


Figure A.38 High timer's criticality: seize in advance

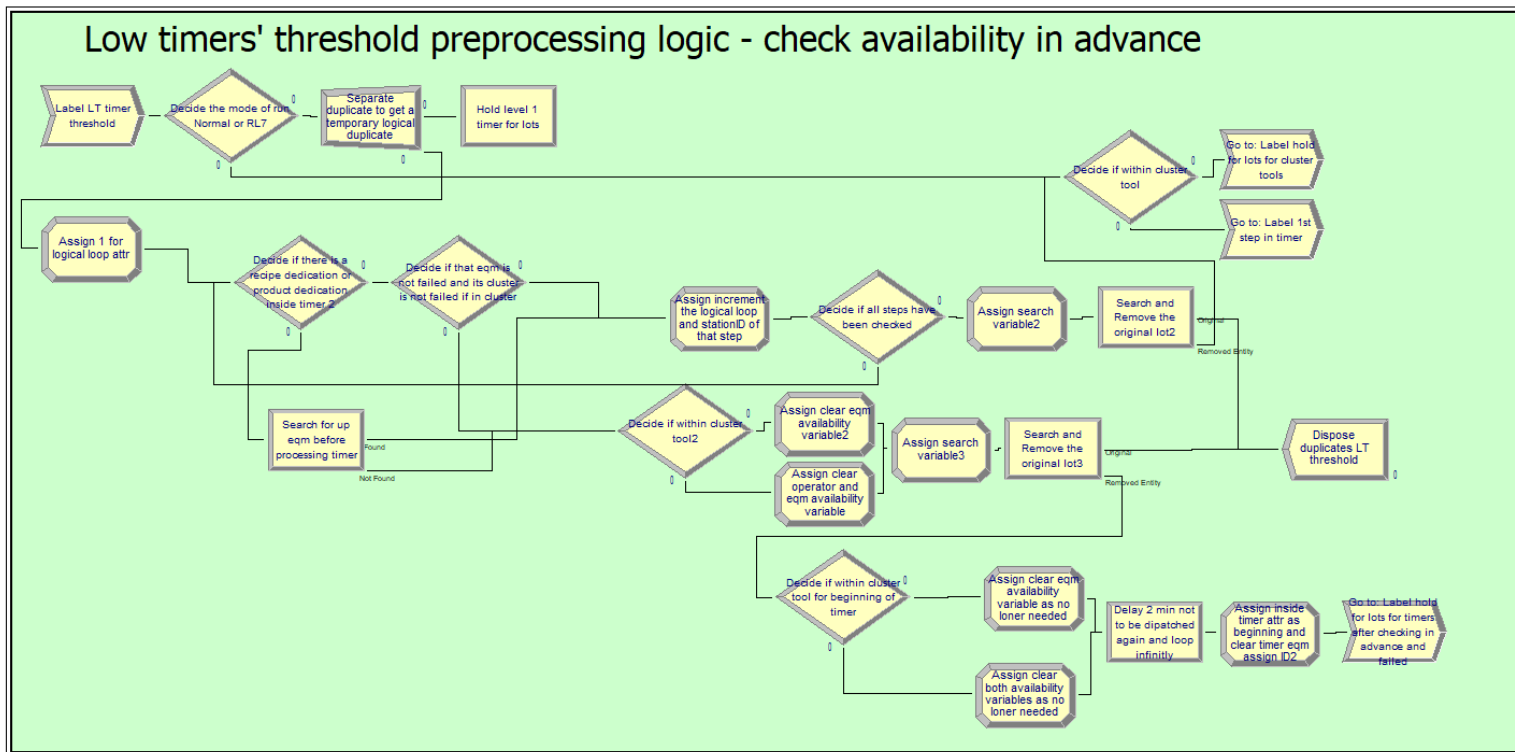


Figure A.39 Low timer's criticality: check availability in advance

Seizing equipment within timer

Figure A.40 shows the group of modules in which lots go through for seizing equipment within their timers' steps. According to whether the lot will start a timer or it is within a timer, and whether equipment are held in advance or not, the lot is routed. Hence, the lot is routed through the model.

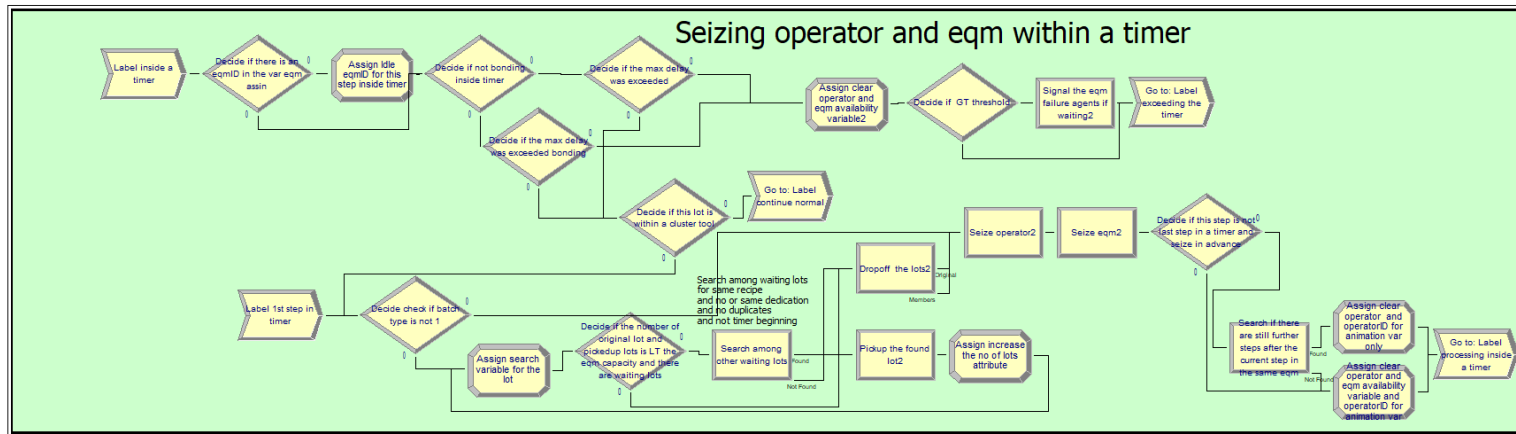


Figure A.40 Seizing equipment within timers

Decisions for timers' limit exceeding

Figure A.41 shows the group of modules in which lots with exceeded timer limit go through to decide, according to a certain probability, whether to scrap or rework the lot. Accordingly, all held equipment are released if any. Also, the reward term for timers is calculated in case of RL dispatching.

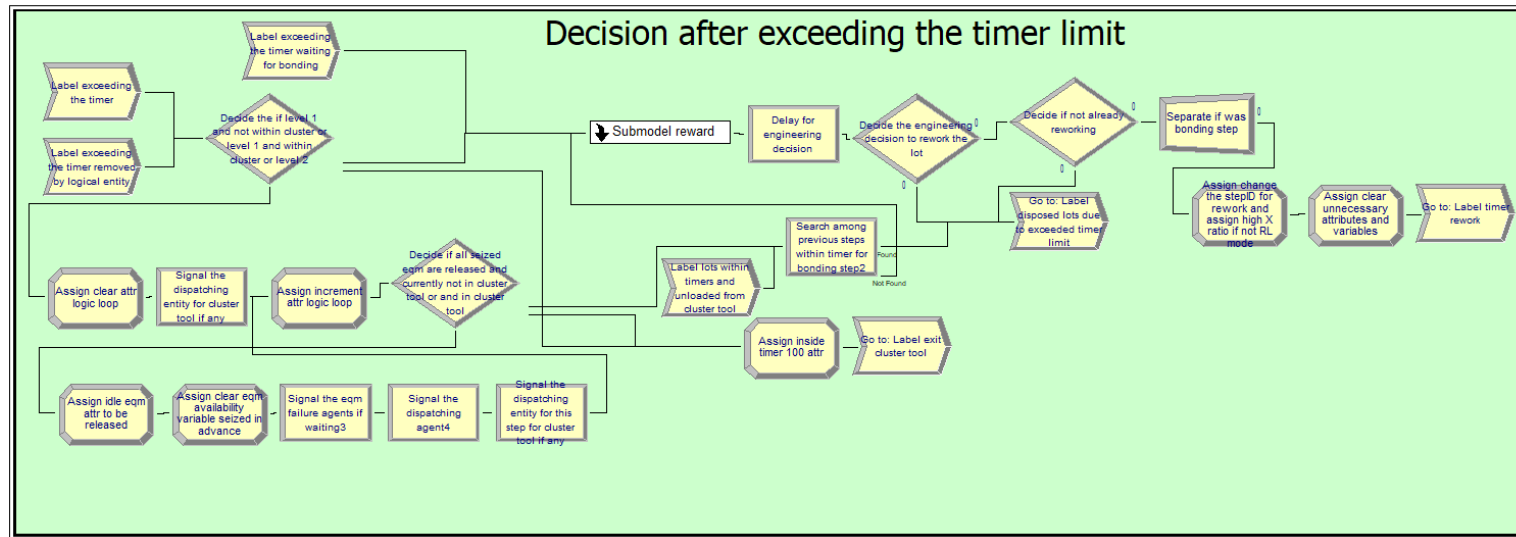


Figure A.41 Exceeding timers' limit

Finished lots leaving the system

Figure A.42 shows the group of modules in which finished lots go through to calculate some statistics, assign some variables, and signal some agents if required.

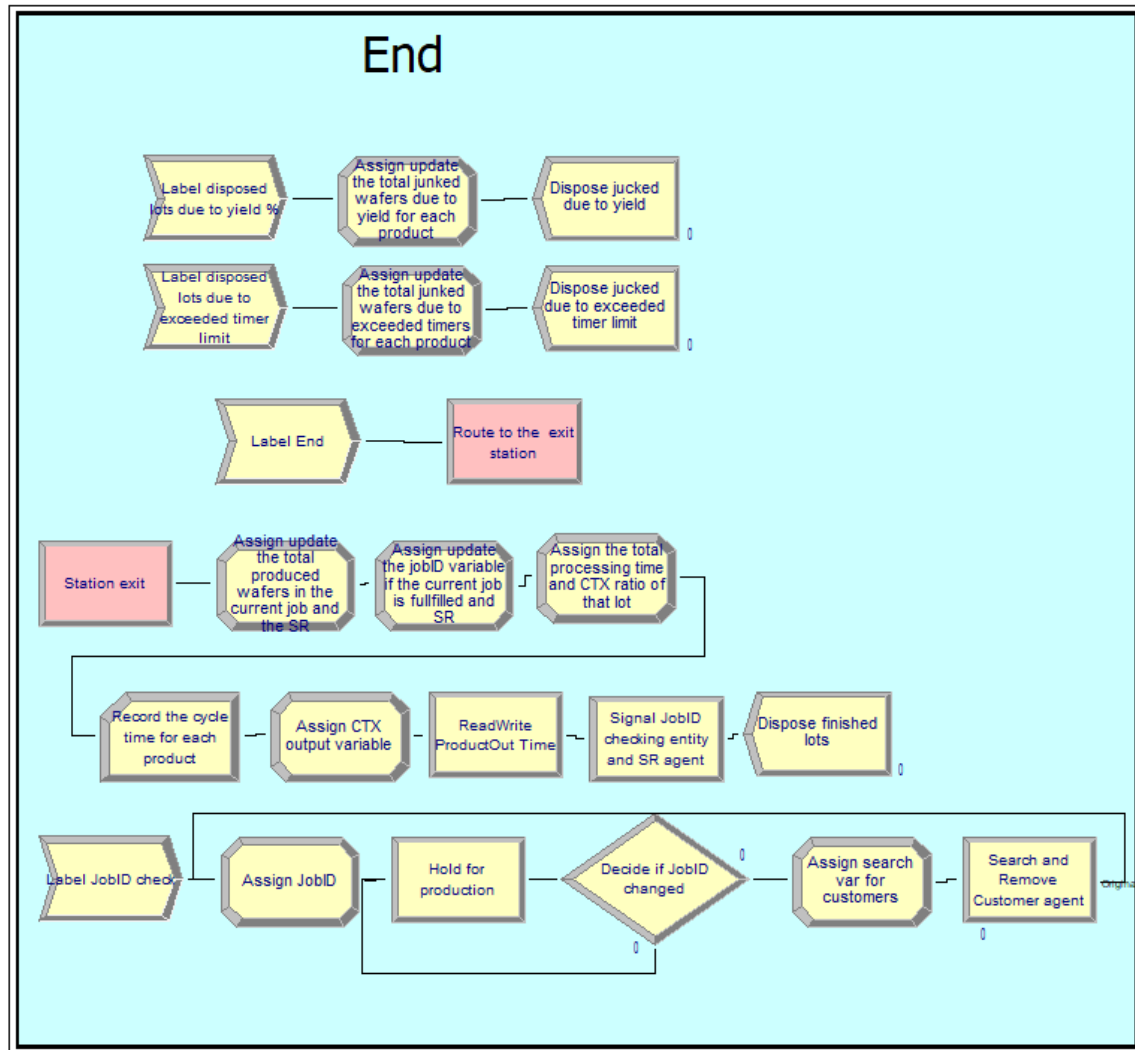


Figure A.42 Exceeding timers' limit