Control of Semiconductor Wafer Fabrication

T. Tolboom

SE 420387

Master's thesis

Supervisor: prof.dr.ir. J.E. Rooda
Coaches: dr.ir. A.A.J. Lefeber

Eindhoven University of Technology
Department of Mechanical Engineering
Systems Engineering Group

Eindhoven, May 2004

FINAL ASSIGNMENT

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mechanical Engineering
Systems Engineering Group

October 2002

Student	T. Tolboom
Supervisor	Prof.dr.ir. J.E. Rooda
Advisor	Dr.ir. A.A.J. Lefeber
Start	October 2002
Finish	October 2003
Title	Control of Semiconductor Wafer Fabrication

Subject

Semiconductor wafer fabrication is one of the most complex manufacturing processes. The complexity is brought by the high-tech processes being applied, their constant development, the multitude of processes and their reoccurring nature.
Due to the high acquisition costs of the equipment, control techniques are used to ensure high utilization of the bottleneck workstations. Scheduling is one of these techniques, requiring a stable and predictable manufacturing environment. Unfortunately, the prerequisite of a stable and predictable environment is rarely satisfied in a loaded wafer fab. In practice, semiconductor wafer fabs are often corrupted by the influence of variability. This causes a certain optimal schedule to become far from optimal. The more variability present in the system, the more often a renewed optimal schedule needs to be determined to deal with the rapidly changing environment.
Up to today there is no unambiguous answer to the question how to control a semiconductor wafer fabrication facility, with a job-shop character, in order to maximize throughput and reduce flow time.

Assignment

Wein (1988) carried out a study concerning the impact control strategies might have on the performance of semiconductor wafer fabrication facilities with a job-shop character. A variety of input control and sequencing rules were evaluated using a simulation model of a representative but fictitious semiconductor wafer fab. Within the following 15 years several new input control and sequencing rules arose, the semiconductor industry evolved and new process generations were introduced.
Investigate wether the conclusions drawn by Wein (1988) are still legitimate within the present situation. Reproduce the simulation results with the use of a model created within the χ-environment. Expand this model to a representative, but fictitious, wafer fab provided with the existing equipment and process generation(s). Expand the number of control strategies evaluated with those mentioned in the literature and develop alternatives for these control strategies. Determine the influence of control strategies on the performance of a semiconductor wafer fabrication facility, with flow time as a possible performance measure. Present an extensive discussion of the results in a report and conclude with recommendations for further research.

Prof.dr.ir. J.E. Rooda

Dr.ir. A.A.J. Lefeber

Systems Engineering

Department of Mechanical Engineering

Preface

This master's thesis reflects the result of my final project of a five year curriculum to attain a Master of Science in Mechanical Engineering with a specialization in Systems Engineering. My 'journey' started in September 1997. After almost four years of study, I joined the System Engineering Group. In October 2001, I was offered the opportunity to conduct my extended internship at MOS4YOU and gain some practical experience. Intrigued by the complex problems of semiconductor manufacturing, I started my final project on the same area of research in October 2002.

This particular project was performed within the 'Optimization and Control' theme of the Systems Engineering Group. The Systems Engineering Group, at the Eindhoven University of Technology, aims to develop methods, techniques and tools for the design and control of advanced industrial systems. The research of the 'Optimization and Control' theme focusses on the development of optimization tools and design of control systems for production systems and machines. This research project focusses on the development of a multi-level production control framework, applicable to a re-entrant (semiconductor) production system.

During my final year project, I was supported by several people, whom I would like to thank here. First of all I would like to thank my coach Erjen Lefeber, for his guidance and support, even in the early hours. Furthermore, I would like to thank professor Rooda for his interest in this research project and my fellow students for their daily support. Especially, I would like to thank the fellow students and staff members of the OptCon meetings, for our constructive discussions on current research within the 'Optimization and Control' theme.

A special word of gratitude goes to my dear friends Ad Kock and Sven Weber, for our interesting discussions, our useless squabbling, but above all for their 'unconditional', much appreciated, support during the various parts of my project.

Last, but certainly not least, I would like to thank my family and friends for patiently supporting me. Especially the love and support of my girlfriend Joanne has kept me going during these tumultuous and sometimes difficult times.

Ted Tolboom

's-Hertogenbosch, May 2004

Summary

So far, no satisfactory control approach for re-entrant production systems has been presented. In spite of the increasing complexity of these production systems, the commonly applied control approaches are mainly based on simple heuristics and operator experience. During the past years, research on re-entrant production control resulted in several theoretical advances, yielding improved accuracy and computational tractability. These findings, however, have not had a lasting impact on the manufacturing floor.

Production control is commonly implemented at various levels of operation. The interaction between the different levels is essential in achieving the goals of the production system. However, research on layered control frameworks that combines approaches at different levels is rare. This research project presents a two layer hierarchical control framework for re-entrant production systems. Different levels are used to subdivide an otherwise intractable control problem into multiple smaller problems (layers). The main objective of this framework is to translate actual demand into controllable events, e.g. dispatching of lots.

The high level control layer computes production targets based on the aggregated state and on the predicted behavior of the production system (Model Predictive Control, MPC). An accurate prediction of the non-linear system dynamics is obtained by deploying resource capacity relations based on approximations from queuing theory ('effective clearing functions'). The parameters required for the definition of the effective clearing functions are determined by an Effective Process Time (EPT) algorithm. A partitioning algorithm is introduced to adequately distribute capacity over lots at different stages of production.

The low level control layer consists of a distributed composite dispatching rule. The use of a composite dispatching rule guarantees a good, yet suboptimal, solution in a short time. This rule determines which lot must be processed (dispatched) next, based on the local state of the production system and the issued production targets. The rule sequences lots based on the difference between the issued due date and the time it could be ready. The lot with the smallest difference (slack) is dispatched.

The performance of the MPC framework is analyzed using a simulation model of a characteristic semiconductor production system. The MPC framework is implemented on two different cases.

The first case encompasses three production steps on three different workstations. This case is used to analyze the performance without the partitioning of capacity over the different production steps. The second case utilizes a similar production system with a re-entrant product flow. In this case, six production steps are performed on workstations identical to those of the first case. The second case is used to analyse the performance of the MPC framework in a re-entrant environment.

To place the performance into perspective, the simulation results are compared to that of the commonly encountered 'Material Resources Planning' (MRP-C) approach. This approach assumes a constant flow time independent of workstation utilization, resulting in a linear relation between production capacity and utilization. Consequently, many standard approaches, such as MRP-C, neglect the influence of variability and overestimate production capacity.

Both cases are subjected to three different simulation experiments. These experiments differ from one another by the enforced demand trajectory. The results of these simulation experiments indicate that the presented MPC framework outperforms MRP-C in the sense that the enforced demand trajectory is followed closely without creating a (permanent) backlog. Both the mean and the variability in the number of backorders are lower for the presented MPC framework than for MRP-C. These results also show that the MRP-C approach is not always able to attain the new demand level on time, introducing a permanent backlog.

The simulation results indicate that the difference in performance increases with increasing utilization. This difference is caused by the fact that MRP-C approximates the resource capacity by two linear constraints. Conversely, the new framework deploys approximations of the actual non-linear behavior. The difference between these two approximations increases with increasing utilization. Consequently, the production targets issued by the MPC framework will better correspond to the actual behavior of the production system at high utilization levels.

Effectively, the reduction in mean and variation of the number of backorders is procured by maintaining a (security) WIP level to counteract the observed variability is the process times. Consequently, a tradeoff will be made with regard to the mean total flow time and the mean WIP level.

In spite of the promising results, several encountered complications, should be resolved.

Samenvatting (in Dutch)

Tot op heden is er nog geen toereikende regelstrategie ontwikkeld voor re-entrant productiesystemen. Ondanks de toenemende complexiteit van productiesystemen, zijn de regelstrategieën die tegenwoordig worden toegepast voornamelijk gebaseerd op eenvoudige heuristieken en op de ervaring van operators. Onderzoek naar regelstrategieën voor re-entrant productiesystemen heeft de laatste jaren verschillende theoretische vooruitgangen opgeleverd, met name wat betreft de nauwkeurigheid en de benodigde rekenkracht. Deze nieuwe bevindingen hebben echter geen blijvende indruk achtergelaten op de fabrieksvloer.

Regeltechniek wordt doorgaans toegepast op verschillende lagen binnen een productiesysteem. De interactie tussen de verschillende lagen is essentieel voor het bereiken van de doelstellingen van het productiesysteem. Desondanks is onderzoek naar regelraamwerken die verschillende strategieën op verschillende lagen met elkaar combineren, schaars. In dit onderzoek wordt daarom een twee-laags hiërarchisch regelraamwerk voor re-entrant productiesystemen gepresenteerd. Meerdere lagen worden toegepast om een anders onhandelbaar probleem op te delen in meerdere kleinere problemen (lagen). Het doel van dit raamwerk is het vertalen van concrete vraag naar regelbare gebeurtenissen, zoals bijvoorbeeld het vrijgeven (starten) van productie.

In de bovenste laag van het regelraamwerk worden de productiedoelen bepaald op basis van de geaggregeerde toestand en het middels een model voorspelde gedrag van het productiesysteem (Model Predictive Control, MPC). Een nauwkeurige voorspelling van de niet-lineaire systeemdynamica wordt verkregen door de capaciteit van een machine of werkstation te baseren op benaderingen uit de wachttijd theorie ('effective clearing functions'). De parameters die nodig zijn voor het definiëren van de 'effective clearing functions' worden bepaald met behulp van een Effectieve Process Tijd (EPT) algoritme. Een algoritme wordt gebruikt om de capaciteit op een juiste manier over de lots, in verschillende stadia van het productieproces, te verdelen.

De onderste laag van het regelraamwerk bestaat uit een samengestelde ordervrijgave regel die gebruik maakt van informatie van het productiesysteem. Het gebruik van een samengestelde regel garandeert een goede, doch suboptimale, oplossing met minimale rekenkracht. Op basis van de locale toestand van het productiesysteem en de afgegeven productiedoelen wordt met behulp van deze regel bepaald welk lot als volgende vrij-

gegeven dient te worden. Deze regel sorteert lots op het verschil tussen het tijdstip waarop een lot klaar moet zijn, en het vroegste tijdstip waarop het klaar kan zijn. Het lot waarvoor dit verschil het kleinste is wordt vrijgegeven.

De prestatie van het gepresenteerde MPC regelraamwerk is onderzocht door gebruik te maken van een simulatiemodel van een representatieve productielijn. Het MPC raamwerk is geïmplementeerd op twee verschillende cases. De productielijn die voor de eerste case gebruikt is bestaat uit drie bewerkingsstappen op drie verschillende werkstations. Deze case is gebruikt om de prestatie van het MPC raamwerk te beoordelen zonder dat capaciteit verdeeld hoeft te worden over de verschillende bewerkingsstappen. De tweede case maakt gebruik van een vergelijkbare productielijn, waarbij de lots echter meerdere malen op hetzelfde werkstation terugkeren (re-entrant productstroom). In deze case worden zes bewerkingsstappen uitgevoerd op dezelfde werkstations als bij de eerste case. De resultaten van deze case zijn gebruikt om de werking van het MPC raamwerk in een re-entrant omgeving te beoordelen.

Om de prestatie van het raamwerk te evalueren, wordt het vergeleken met de prestatie van de veel toegepaste 'Manufacturing Resources Planning' (MRP-C) strategie. Deze strategie gaat uit van een constante bewerkingstijd, ongeacht de bezettingsgraad van de machine of werkstation. Deze aanname impliceert een lineaire relatie tussen productiecapaciteit en bezettingsgraad. Door deze aanname wordt de invloed van variabiliteit verwaarloosd, waardoor de productiecapaciteit overschat wordt.

Simulatie experimenten duiden erop dat het MPC raamwerk beter presteert dan MRP-C wanneer gelet wordt op het correct volgen van de opgelegde vraag zonder permanent productietekort (of productieoverschot). Zowel het gemiddelde van, als de variabiliteit in, het productietekort is lager voor het MPC raamwerk dan voor MRP-C. Voorts blijkt MRP-C niet altijd in staat is om op tijd aan de nieuwe vraag te voldoen, waardoor een permanent productietekort ontstaat.

De simulatie experiments geven daarnaast aan dat het verschil in prestatie, tussen de twee strategieën, toeneemt bij toenemende bezettingsgraad. Dit kan verklaard worden doordat MRP-C de machine capaciteit benaderd met twee lineaire constraints. Het gepresenteerde MPC raamwerk maakt daarentegen gebruik van een benadering van het werkelijke niet-lineaire gedrag. Aangezien het verschil tussen beide benaderingen toeneemt met toenemende bezettingsgraad, zullen de door het MPC raamwerk vrijgegeven productiedoelen, bij een hoge bezettingsgraad, beter overeenkomen met het werkelijke gedrag van het productiesysteem.

Het behaalde resultaat - het reduceren van zowel het gemiddelde van als de variabiliteit in het productietekort - wordt bewerkstelligd door het aanhouden van een veiligheidsvoorraad dat de geobserveerde variabiliteit in de bewerkingstijden moet compenseren. Hierdoor wordt echter een concessie gedaan met betrekking tot de gemiddelde totale doorlooptijd en het gemiddelde aantal lots dat in het productiesysteem verblijft.

Ondanks de veelbelovende resultaten moeten enkele ondervonden complicaties nog verholpen worden.

Contents

List of Definitions

Clearing function	Expression for the expected output of (a part of) the production system as a function of the WIP over a given period of time
Condition blocking	All types of blocking that occur since a certain condition on the resources is not (yet) met.
Dispatched work	Set of lots (work) released into the production system
Dispatching	Determine which lot to process next, at the moment a resource becomes idle
Effective clearing function	Clearing function based on exact relations or on approximations from queuing theory
Effective Process Time	The total time seen (claimed or consumed) by a lot on a workstation
EPT realization	The time a lot was in process plus the time a lot (not necessarily the same lot) could have been in process
Estimated clearing function	Clearing function based (fitted) on historical data
Planning	Allocation of (forecasted) demand to available resource capacity
Possible work	Set of lots (work) that are allowed to be processed, with respect to the issued production targets
Re-entrant production system	Production systems where lots may return more than once to the same resource for repeated stages of processing
Scheduling	Allocation of a specified lot process action to a resource at a specified time segment
Sequencing	Sorting of the available work with respect to a predefined algorithm

List of Acronyms and Symbols

Acronyms

BOM	Bill of Materials
BOR	Bill of Resources
CONWIP	CONstant Work-In-Process
CRP	Capacity Requirements Planning
DES	Discrete Event System
DESM	Discrete Event Simulation Model
EDD	Earliest Due Date
EPT	Effective Process Time
FIFO	First-In-First-Out
FRC	Flow Rate Control
FS	Fluctuation Smoothing
LDM	Linear Discrete Model
LP	Linear Programming
LS	Least Slack
LS/n	Least Slack per remaining process step
LSC	Least Setup Cost
MPC	Model Predictive Control
MPS	Master Production Schedule
MRP-I	Material Requirements Planning
MRP-II	Manufacturing Resources Planning
MRP-C	Capacitated MRP
mSLM	multiple Single Lot Machine
RCCP	Rough Cut Capacity Planning
RHS	Rolling Horizon Scheme
SA	Starvation Avoidance
SB	Shifting Bottleneck
SBH	Shifting Bottleneck Heuristic
SCLP	Separated Continuous Linear Program
SLQ	Stochastic Linear Quadratic
SRPT	Shortest Remaining Process Time

VPP	Variable Priority Policy
WIP	Work-In-Process
WR	Workload Regulating
WSPT	Weighted Shortest Process Time

Symbols

c	Linearization index
i	Lot index
j	Server index
k	Workstation index
l	Step index
$\star$	Point of linearization
$\boldsymbol{AA}_{i,j,k}$	Actual arrival of lot i on server j of workstation k
$\boldsymbol{AD}_{i,j,k}$	Actual departure of lot i on server j of workstation k
b_l	Batch size of step l
BO_t	Backorders at the end of period t
c_0^2	Squared coefficient of variation of the clean process time
c_a^2	Squared coefficient of variation of the inter arrival time
c_d^2	Squared coefficient of variation of the inter departure time
c_e^2	Squared coefficient of variation of the effective process time
d_i	Due date of lot i
d_t	Demand for lots, during period t
D_{ini}	Initial period of the demand trajectory, prior to the demand transition
D_{max}	Demand value after demand transition
D_{min}	Demand value before demand transition
D_{per}	Transition period length of the demand trajectory
D_t	Actual departures during period t
f	Clearing function
g	Column vector defining cost allocation in the LDM
I_t	Inventory of finished lots at the end of period t
$\mathbf{lb}$	Lower bound of linear discrete model variables $\mathbf{x}$
m	Number of (identical) parallel servers of a workstation
n_k	Number of process steps for workstation k
N	Total number of process steps
N_k	Total number of EPT realizations on workstation k
$\boldsymbol{OR}_{i,j,k}$	Time instance of order-release of lot i on server j of workstation k
R_t	Actual releases during period t
s_i	Slack of lot i
s_k	Physical storage capacity of workstation k
$\mathcal{S}_k$	Set of steps belonging to workstation k
t_0	Mean clean process time

$t_{0,1^{st}}$	Mean clean process time of the first step
$t_{0,2^{nd}}$	Mean clean process time of the second step
t_e	Mean effective process time
$t_{h,k}$	Lot handling (load/unload) time for workstation k
u	Utilization
$\mathbf{ub}$	Upper bound of linear discrete model variables $\mathbf{x}$
w	Work-in-process
w_n	Weight factor for EPT realization Y_n
$\bar{w}_t$	Average WIP during period t
$W_{l,t}$	WIP of step l, at the end of period t
$\mathbf{x}$	Column vector with the Linear Discrete Model variables
$X_{l,t}$	Production quantity of step l, during period t
Y_n	The nth EPT realization
α	(fixed) clearing factor
$\alpha(w)$	Non-linear clearing function
$\alpha^c_{k,t}$	c-th gradient parameter for workstation k during period t
$\boldsymbol{\alpha}$	Set of linear gradient parameters
β_i	Least slack parameter
$\beta^c_{k,t}$	c-th scaling parameter for workstation k during period t
$\boldsymbol{\beta}$	Set of linear scaling parameters
γ_i	Least slack parameter
δ	Throughput
δ_{max}	Maximum throughput of a resource
Δ_0	Offset of the clean process time distribution
ζ_l	Estimate of the remaining flow time from process step l
ι	Inventory holding cost
λ	Arrival rate
o	Backorder cost
σ^2_e	Variance of the mean effective process time t_e
τ	Time of occurrence
$\tau^f_{i,j,k}$	End time of the EPT realization of lot i on server j of workstation k
$\tau^s_{i,j,k}$	Start time of the EPT realization of lot i on server j of workstation k
φ_b	Mean wait-in-batch-time at a batching workstation
φ_q	Mean queuing time at a workstation
φ	Mean flow time
ω_l	Holding cost of step l

Chapter 1

Introduction

Semiconductor wafer fabrication is one of the most complex, competitive and capital intensive production processes. The complexity is brought about by, the number of process steps and the product-specific re-entrant flow across a number of unique tool groups (multiple, identical machines operating in parallel), some of which process jobs in batches, while other require sequence-dependent setups. The re-entrant behavior along the most prominent tool groups within a semiconductor production facility is illustrated by Figure 1.1. Here, the lithography tool group forms the pivot of the facility.

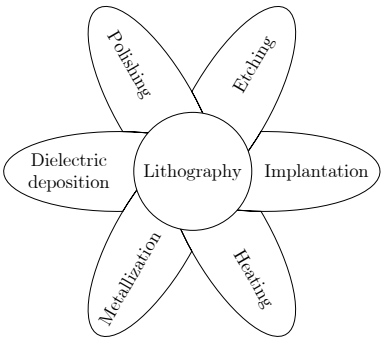


Figure 1.1: Re-entrant nature of semiconductor manufacturing (Campen, van 2001)

The semiconductor industry has experienced a tumultuous development in the last decade and has become more competitive. Different control approaches are deployed to maintain a high throughput, low total flow time (time to process a lot from start to completion) and high on-time delivery performance. These control approaches significantly affect the performance of the production process.

In spite of the importance of this control process, applied control approaches have not changed substantially throughout the years. Though there has been important theoretical advances, much research in this field has not had a lasting impact on the manufacturing floor (Uzsoy et al. 1992).

Whether implemented in home-grown spreadsheets or commercial software packages, current production control is based on simple strategies obtained on the basis of (operator) experience, trial-and-error, simplified static models and heuristics.

Control is commonly deployed at various levels of production system, ranging from (short-term) tool level to (long-term) aggregate production planning. In general, these control levels operate separately without interaction. However, the interaction between different levels is essential in achieving the goals of the production system, especially in complex production environments with a high degree of variability. Most research on control of re-entrant lines focusses on one specific level. Control of multiple levels has mostly been neglected.

Objective

So far, no satisfactory control framework for re-entrant production facilities has been presented. The goal of this research is to develop such a framework. The framework should be able to cope with variability, it should be conceptually simple and it should be applicable to a full-scale production environment.

The performance of the framework should be analyzed by comparing its performance to that of several widely used approaches.

Approach and outline

In order to meet the objective, the following approach, which also defines the outline of this report, is used.

Chapter 2 surveys several control approaches encountered in literature. The chapter starts with a decomposition of control approaches into three levels: high-level, intermediate-level and low-level. Different control approaches will be reviewed with respect to the decomposition.

Subsequently, in Chapter 3 a two layer hierarchical control framework is presented. The general structure and objectives of the control framework are presented. Next, an in-depth description of the control approaches incorporated in the two layers is provided.

In Chapter 4, two different cases are presented. They will be used to evaluate the performance of the presented control framework. Both cases are based on the 'Intel Case' (Kempf 2003) and feature characteristics commonly encountered in a re-entrant semiconductor production line.

In Chapter 5, first a set of simulation experiments is defined that will be used to evaluate the performance of the control framework. The simulation results of the control framework with respect to the two cases defined in Chapter 4 are discussed.

To place the performance of the control framework into perspective, the simulation results are compared with the results of a production system controlled by the 'traditional' MRP-C approach (Tardif and Spearman 1997).

Finally, in Chapters 6 and 7, conclusions of this research are drawn and recommendations for future research are made.

Chapter 2

Literature survey

A characteristic feature of re-entrant production systems (Kumar 1993) is the competition between lots, each at a different stage of production, to claim capacity of the same machine. This feature complicates the control of re-entrant production systems. Typical application area of re-entrant systems is the semiconductor industry, with a tumultuous development in the past two decades. Consequently, control of re-entrant systems has become a relevant topic in the past decades.

Uzsoy et al. (1992, 1994) present an extensive review on various control strategies at different levels of the (re-entrant) semiconductor production process. The subsequent sections of this chapter provide an overview of significant contributions to re-entrant production control systems.

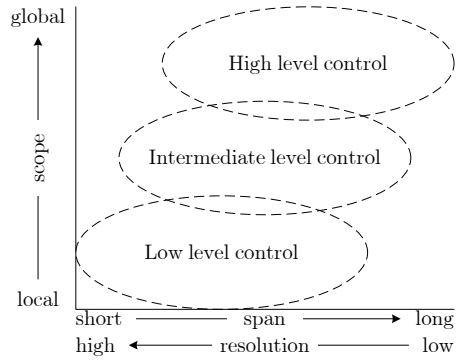


Figure 2.1: Distinguishing characteristics of control decomposition

In literature, control systems are generally decomposed according to their natural hierarchical structure. Rather than opting for a decomposition based on the organizational structure, emphasis, within this thesis, is placed on the decisional (i.e. span, scope and resolution) aspects of control systems (Weber 2003). The use of a high-level, intermediate-level and low-level control is advocated as a governing decomposition, illustrated by Figure 2.1.

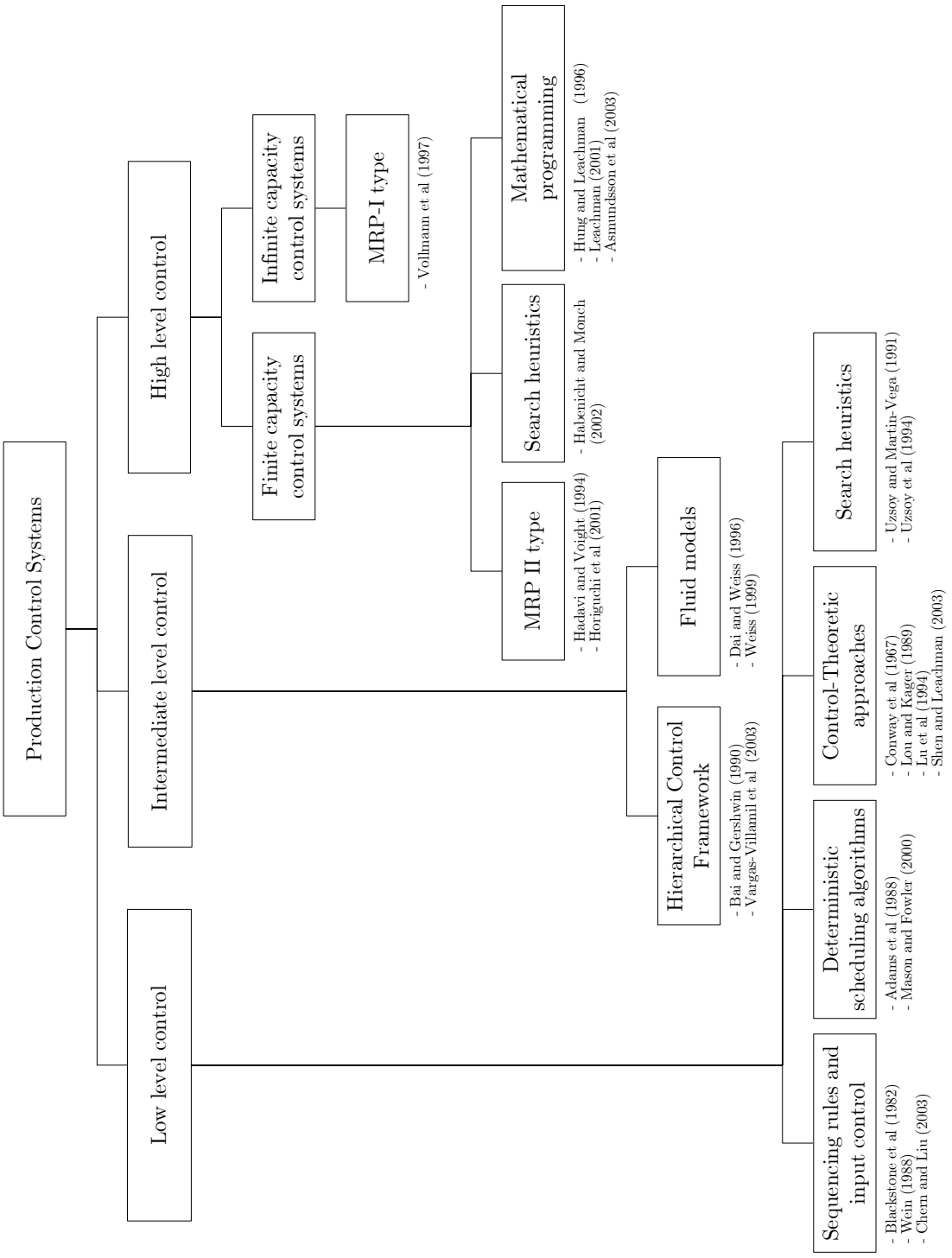


Figure 2.2: Decomposition of production control.

Here, high level control focusses on global, system wide, optimization of the aggregated production system (i.e. long horizon, low resolution) whereas low level control aims at short term, local optimization of the production system (i.e. short horizon, high resolution). Intuitively, intermediate level control forms the grey area in between.

Another distinguishing characteristic between control approaches is whether they can be applied on-line or off-line. Both at high and low level on-line approaches can be deployed, i.e. the time required to optimally solve the control problem is small compared to the resolution. Typical example of such approaches are sequencing rules (Uzsoy et al. 1994). Other approaches are forced to solve the optimization problem off-line, due to the involved computational effort. An example of off-line deployment is the Rolling Horizon Scheme (Toba 2000). The Rolling Horizon Scheme (RHS) solves its control problem, with a limited time horizon, using up-to-date data, at a fixed rescheduling interval. The most recent schedule is then applied to the underlying production system.

In the following sections, several commonly encountered production control approaches will be described briefly. The used decomposition, supplemented with respective references, is illustrated by Figure 2.2.

2.1 High level control

Typically high level, long-term, production control systems translate predefined goals into meaningful assignments for underlying controller(s), based on the aggregated state of the system. These control approaches can be characterized according to their assumed relationships between throughput (δ) and work in process (w); the finite or infinite capacity relation.

Infinite capacity control systems, such as Material Requirements Planning, MRP-I (Graves 1985, Vollmann et al. 1997), and some mathematical programming approaches (e.g. Johnson and Montgomery 1974) assume constant flow times, independent of the workstation utilization. According to Little's law (Little 1961), these control systems implicitly assume a linear relation between δ and w, reflected by Figure 2.3.

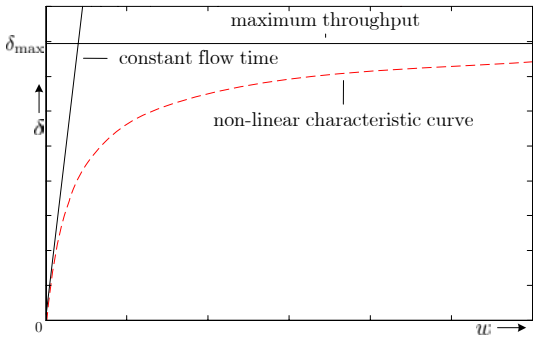


Figure 2.3: Relation between δ and w for a $G/G/2$-workstation.

MRP-I generates a Master Production Schedule (MPS), containing a detailed schedule of individual jobs, based on the Bill of Materials (BOM) for each (sub)product and (fixed) lead times to satisfy demand estimates. Though conceptually simple, scheduling all jobs is still computationally demanding. Therefore, many MRP-I systems tradeoff accuracy for computational tractability by using discrete time-buckets (intervals) and linearized (capacity-) constraints. However since production planning is the task of determining factory load and product mix over time, it can be difficult to ascertain the validity of a production plan derived using constant flow times.

Finite capacity systems, on the other hand, explicitly model the relationship between δ and w, although they may do so in different levels of detail. Models incorporating a maximum throughput rate δ_{max} (Figure 2.3) obtain the lowest level of detail. Issue with this relation is that it implies instant production.

Supported by both academic literature (Hung and Leachman 1996, Horiguchi et al. 2001) and industrial practice, the characteristic relationship between δ and w is assumed to be highly non-linear, owing to the queueing dynamics involved. Therefore, accurate system dynamics is obtained by models describing the 'characteristic' behavior between δ and w, reflected by Figure 2.3. In queueing theory this relation is described by the formula of Pollaczek–Khinchin (Pollaczek 1930).

A common used subdivision of the finite capacity high level control class is based on the type of (optimization) approaches used (Uzsoy et al. 1994, Toba 2000, Habenicht and Mönch 2002). Subclasses are formed by approaches deploying Manufacturing Resources Planning (MRP-II) type solution approaches, search heuristics and mathematical programming to solve the optimization problem. The presented subclasses, together with several commonly used approaches, will be described in more detail in the subsequent subsections.

MRP-II type

Unlike the original MRP-I type approaches, MRP-II type approaches incorporates finite-capacity modules like 'Rough Cut Capacity Planning' (RCCP) and 'Capacity Requirements Planning' (CRP). Both modules work with the principle of capacity-feasibility; enforcing a maximum throughput rate (recall Figure 2.3) on the resources.

The work of Horiguchi et al. (2001) is an example of an MRP-II system with RCCP module, applied to a re-entrant production system. Within the RCCP module, the time required to produce a particular end product at workstations is determined based on the Bill Of Resources (BOR). Jobs are allocated to a time-bucket only if sufficient capacity is available according to the MPS. In Horiguchi et al.'s (2001) approach, only the available capacity of the (near-)bottleneck stations is considered explicitly while developing the plan. Intuitively this yields a finite-capacity plan assuming all other stations have infinite capacity.

The approach is driven by the need to trade off the increased solution accuracy of a finite-capacity model to the increased computational time required to obtain capacity feasible plans at different stations.

Both MRP-I and MRP-II are implemented in many commercial software packages used in semiconductor manufacturing. An example of industrial applications of MRP-II type systems is ReDS (Hadavi and Voigt 1994).

Search heuristics

The second subclass of finite capacity high level control consists of approaches using search heuristics to solve the optimization problem. Examples of commonly used algorithms are the 'branch and bound' and 'beam search' algorithms. In essence, a branch and bound-algorithm creates branches by selecting a partial schedule and computing lower limits (bounds) based on the total flow time. If the bound on a branch exceeds the total flowtime of the best (complete) schedule found so far, it is no longer considered. The beam search algorithm is a derivative of the branch and bound- algorithm . However instead of checking each branch, the beam search algorithm checks only branches that satisfy predefined criteria. Although solution speed is increased, obtaining an optimal solution is not guaranteed.

Fargher and Smith (1994) and Fargher et al. (1994) use a beam search-algorithm in combination with backtracking steps for lot release and for determination of schedules. Habenicht and Mönch (2002) developed a finite-capacity beam search-algorithm, to determine the start and end dates of the jobs. The algorithm ranks the lots by their importance (outer loop) and uses a beam search-algorithm to determine the start and end dates of the lot (inner loop). Due to the existence of parallel tools, the capacity of the tool groups is reviewed as the sum of the individual tool capacities. The proposed algorithm is considered for integration in a more general framework or distributed hierarchical production control (Vargas-Villamil and Rivera 2001).

Mathematical programming

Many production control systems are based on optimization approaches that deploy mathematical programming to determine production parameters. A large variety of optimization goals is available, examples are; achieve improved short-term production rates of all products at each equipment unit (Gershwin et al. 1985), cost minimum production rates of each product (Bai et al. 1990) and processing operations at each equipment unit to minimize inventory (Liao et al. 1996).

Hung and Leachman (1996) proposed an alternative for the prevailing use of the fixed lead time assumption many high level control systems use. They introduce an approach for automated control based on iterative Linear Programming (LP) optimization and discrete event simulations (Figure 2.4).

The goal of this approach is to plan rates of product release over a varying time horizon. Discrete event simulation is used to approximate the characteristic relation between w and δ as shown in Figure 2.3. Iteration continues until a satisfactory agreement between simulation and LP models is obtained.

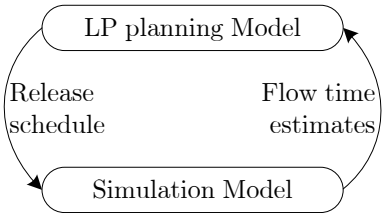


Figure 2.4: Iterative scheme to obtain approximated system dynamics

Leachman (2001) proposes an alternative to the use of discrete event simulation calculations to approximate the characteristic relation. If many products are to be processed, it is impractical to formulate a model incorporating variables and constraints for each process step. Instead, Leachman (2001) formulates a model including only variables representing the release of raw material for processing each product. Flow time parameters are deployed to estimate subsequent workloads associated with the release of raw material at each downstream process step. The model includes capacity constraints for multiple resource types and inventory balance constraints on completed products.

Another approach is used by Asmundsson et al. (2003). Here, an accurate capacity constraint within a mathematical programming model is introduced via non-linear clearing functions (Asmundsson et al. 2003). These functions enable accurately capturing of the characteristic non-linear lead time dynamics, into a 'text-book' mathematical programming model. Based on a combination of commonly used approximations from queuing theory and empirical data obtained from either simulation models or historical data, the approach is implemented into a small scale mathematical model. Simulation results show that the approach yields computational tractable and accurate results.

2.2 Low level control

Low level control of a production process aims at local optimization based on the short-term state of the system. A state–of–the–art review of low level control, applied to the semiconductor production process, is presented by Fowler et al. (2002).

Research on low level control is commonly decomposed into four different classes. The different subclasses are formed by 'sequencing rules and input control', 'deterministic scheduling algorithms', 'control-theoretic approaches' and approaches deploying 'search heuristics'. The presented subclasses, together with several commonly encountered approaches, will be described in more detail in the subsequent subsections.

Sequencing rules and input control

Sequencing rules and input control form the major contribution to low level control systems and are widely used in commercially available control systems. These approaches are used to determine the optimal sequence of lots (to be processed next) and the optimal moment of lot release into the facility.

Scheduling all jobs on all machines is hard, both from a theoretical and practical point of view. The computational cost involved in scheduling a semiconductor production facility are too high. Furthermore, in a highly dynamic environment, a schedule is practically obsolete at the moment of release. A traditional alternative to scheduling, that is commonly applied to the semiconductor shop-floor, is the use of local sequencing rules. These rules are used to determine the optimal job to process next, at the moment a workstation becomes available.

A characteristic feature of sequencing rules is their myopic nature; sequencing rules review only the local state of the workstation, sometimes including pre-defined bottleneck stations. Since the choice on what to process next at a certain workstation significantly influences the performance of the downstream workstations, sequencing rules are not likely to obtain a global optimum.

Many different sequencing rules have been proposed by researchers as well as practitioners. Blackstone et al. (1982) and Wein (1988) provide a survey of such rules. Commonly used sequencing rules include: First-In, First-Out (FIFO), Earliest Due Date (EDD), Weighted Shortest Process Time (WSPT), Shortest Remaining Process Time (SRPT), Least Slack (LS), and Least Setup Cost (LSC). More complex dispatching rules are essentially an amalgam of the constructs mentioned above in the form of ratios, truncated forms, composite dispatching rules, conditional combinations, or multilevel rules.

In complex, re-entrant, production systems, simple sequencing rules like First-In-First-Out (FIFO) do not function well. Alternatives that have proven to work well are the Shortest remaining Process Time (SPT) or Earliest Due Date (EDD) rules. Both with different objectives, respectively reducing the average total flow time and the average tardiness. Note that the performance of the sequencing rules under observation depends highly on the objective of the rule.

Several sequencing rules are designed for case specific application. An example is a set of family-based sequencing rules (Chern and Liu 2003), introducing positive discrimination for lots of the same family (type). Goal of the family-based sequencing rules is to reduce the unnecessary setup times for systems with multiple (product-)types.

Sequencing rules are usually complemented by some form of input control. These approaches aim to determine the optimal moment to release lots into the facility and are based on the observation that an increase of work in progress results in increased mean flow times (Little's law). These types of policies attempt to achieve shorter, more predictable flow times by releasing work in a controlled manner.

Traditionally, input control can be classified into push and pull orientated philosophies. Coarsely put, push systems schedule the release of work based on the (forecasted) demand, whereas pull systems authorize the release of work based on the system status (upstream information) (Hopp and Spearman 2000). Within complex production systems, such as highly re-entrant semiconductor production lines, the choice between push or pull is not always straight forward (Krishnamurthi and Suri 2000).

Falling between the pure push and pull philosophy, the robust CONWIP (constant WIP level) control policy (Spearman et al. 1990) focusses on WIP control and requires an understanding of the characteristic non-linear relation between w and δ (Figure 2.3). New lots are released in a pull manner, machines start process in a push manner. CONWIP attempts to keep the WIP level on target, which is set according to the desired throughput rate.

Wein (1988) examined, through the use of a simulation model, four different input control approaches (random and uniform interarrival times, CONWIP and a Bottleneck approach) combined with several sequencing rules. The Workload Regulating (WR) policy, introduced by Wein (1988), tends to release a lot into the system whenever the total amount of remaining work in front of a bottleneck station falls below a predescribed level. Goal of this policy is to reduce both mean and variance of flow time.

Glassey and Resende (1988) developed an input control policy similar to WR; the Starvation Avoidance (SA) policy. This policy focusses on a single bottleneck station and determines a virtual inventory measured over a lead time, which is used to regulate work release. SA attempts to release work into the fab in such a manner that it will arrive at the bottleneck just before it could become idle (starve). Glassey and Petrakian (1989) complement the Starvation Avoidance policy with a new sequencing rule to increase effectiveness of SA. Furthermore, SA has been expanded for multiple bottlenecks systems by Leachman et al. (1988).

Simulation results of both Wein (1988) and Glassey and Resende (1988) indicate that the use of sequencing and input control rules have a significant impact on the average total flow time. The performance of the sequencing rules used dependents on both the number of bottleneck stations and the type of input control used.

Deterministic scheduling algorithms

Deterministic scheduling algorithms are based on the deterministic scheduling paradigm. These algorithms assume discrete, deterministic data, known *a priori*.

The Shifting Bottleneck (SB) algorithm (Adams et al. 1988) was developed for the problem of minimizing makespan in a job shop where the sequence of machines to be visited is known in advance. The SB approach is an approximation approach to the general job shop problem, which proceeds by optimally solving a sequence of single–machine problems using a branch and bound-algorithm.

First the problem (facility) is divided into several subproblems (workstations). A disjunctive graph representation is used to capture the interactions between the subproblems. Subsequently, each subproblem is scheduled and ranked in order of criticality, based on a predefined performance measure. Iterative rescheduling, based on updated rankings, continues until all subproblems have been scheduled.

Similar to the original SB algorithm, the Shifting Bottleneck Heuristic (SBH) (Mason and Fowler 2000) deploys a 'divide and conquer' approach by decomposing the wafer fab scheduling problem into smaller, more tractable subproblems. A difference with the original SB algorithm is the dependency upon the type of tool group being evaluated (e.g. single lot, multiple lot and batch machines). Different approaches are applied during the solution of the tool group subproblems.

Control-Theoretic approaches

Concepts from optimal control theory have been used to develop sequencing rules and input regulating rules for semiconductor fabrication facilities.

Lou and Kager (1989) presented an order release strategy called Flow Rate Control (FRC) to reduce WIP and improve due date performance. Here, FRC is based on a two-region control system model that either ceases input or introduces orders at the maximum rate based on a calculation of surplus inventory. The novel suggestion of FRC is the decomposition of the re-entrant flow process into a strictly serial production line using a virtual buffer concept, illustrated by Figure 2.5. An interesting feature is that, instead of using only local information at the workstations, more global information is taken into account by calculating surplus for a workstation as the summation of the downstream inventories.

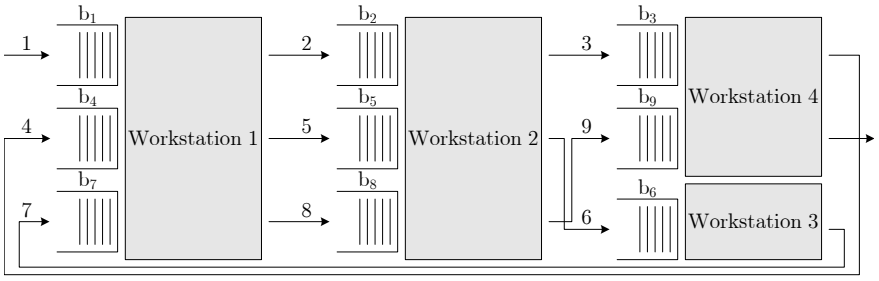


Figure 2.5: A re-entrant flow line decomposed into a virtual buffer (serial) flow line.

Other approaches build on the virtual buffer concept and use e.g. the Least slack (LS) policy (Conway et al. 1967). The Least Slack (LS) policy assigns highest priority to a lot i whose slack s_i, calculated by (2.1), is smallest. Whenever a machine becomes idle, it scans the buffers and chooses lot i, with smallest s_i for processing.

$$s_i = \beta_i - \gamma_i \tag{2.1}$$

The values for parameters β_i and γ_i depend on the optimization goal. If the goal is to reduce the variance of the lateness of all lots, β_i is equal to the due date of lot i and γ_i is equal to an estimate of the amount of time remaining until the lot is completed. Lu and Kumar (1991) and Kumar (1993) have proven that the entire class of LS policies are stable in a deterministic setting.

A subset of the Least Slack policy are the Fluctuation Smoothing (FS) policies (Lu et al. 1994), featuring particular choices for β_i and γ_i. If the goal would be to minimize the mean and variance of the cycle time, slack is estimated by $n/\lambda - \gamma_i$ for the n-th lot to enter the fab. Here, λ denotes the arrival rate of lots to the fab. The delay estimate γ_i is estimated through the use of iterative simulation. The fluctuation smoothing policy is expanded for multi-process systems by Sohl and Kumar (1995).

Only a few policies have been developed to deal with the situation of demand, yield and (resource-)capacity uncertainty, which are persistent in most semiconductor facilities. Shen and Leachman (2003) use a solution methodology based on Stochastic Linear Quadratic (SLQ) optimal control theory. Based on the paradigm of SLQ, the proposed methodology balances WIP flows with respect to production targets in the presence of uncertainties, for a small theoretical case inspired by wafer manufacturing application. Although the approach seems promising, Shen and Leachman (2003) make recommendations for future research to test the SLQ methodology in a more general production network.

Search heuristics

The last class of low level control approaches deploy search heuristics to generate detailed optimal and feasible production schedules. Although the search heuristics, implemented at both low level and high level, are comparatively the same, the level of detail differs. At high level, products are allocated to available workstation capacity (planning), whereas at low level individual operations (jobs) are allocated to a particular machine at a particular time frame (scheduling).

Examples of applied low level search heuristics are the branch and bound and beam search as described earlier, as well as heuristic neighborhood search (Uzsoy et al. 1991). The goal of the latter is to minimize the average maximum lateness of each product. An overview of control approaches for re-entrant flow lines is presented by Uzsoy et al. (1994).

2.3 Intermediate level control

The intermediate level control class forms a grey area between low and high level control. Within this thesis the intermediate level will encompass two types of control policies; hierarchical control frameworks and fluid models, which are briefly described in the following subsections.

Hierarchical control framework

Hierarchical control frameworks typically use a naturally hierarchical structure, as described in the beginning of this chapter, for the decomposition of control levels (Gershwin 1989, Sethi and Zhang 1994). Within a production facility, the frequency of events is 'low' at the highest level and 'high' at the lowest level. At the highest level, targets are determined based on economic factors and partial state information. These targets are passed on to the lower level controller which ensures that the appropriate decisions are made, based on the state of the factory, so that the desired targets are achieved to the best possible extend. Ultimately, it is the low level controller that decides how resources are allocated within the production facility. The framework is illustrated by Figure 2.6.

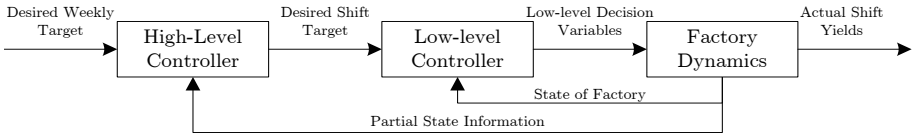


Figure 2.6: Hierarchical control framework

Tsakalis et al. (1997) describes issues concerning hierarchical modeling and control within a mini-fab based on the time scale theory of Sethi and Zhang (1994) and Gershwin (1989). At the high level controller, low resolution flow models are used based on average variables, e.g. average cycle times, of the underlying system. Whereas the low level controller encompasses a high resolution model requiring more data, often obtained by sampling the original discrete event system.

Another example of a hierarchical control framework, developed for a (re-entrant) semiconductor wafer fab, is presented by Bai et al. (1990). This framework uses hierarchical decomposition and control-theoretic approaches to control flexible production systems. The objective is to meet defined throughput goals. The approach is based on classifying the events that take place in a wafer fab, using the frequency of occurrence and the fact that they are controllable.

Vargas-Villamil and Rivera (2001) developed a three layer hierarchical framework which relies on the use of Model Predictive Control (MPC), online parameter estimation and a distributed control policy. At the top level, the parameters of the underlying discrete event system are obtained on-line ('parameter estimation layer'). In the intermediate level an l_1-norm MPC, which uses a discrete linear model, addresses the long-term inventory and production control problem ('optimization layer'). The rolling horizon feature of MPC allows the algorithm to simultaneously act as a long-term optimizer and as a controller. At the lower level, a variable priority policy (VPP) is used to make the appropriate discrete event decisions, based on the desired aggregated targets issued by the optimizer ('direct control layer').

Vargas-Villamil et al. (2003) applied the framework to a discrete event model of a small scale re-entrant production line, consisting of five-machines and six process steps, developed by the Intel Corporation (Kempf 2003). Although results from this discrete event model seem promising, many questions remain to be answered by future research.

Fluid models

A fluid model can be regarded as a continuous representation of the discrete stochastic queueing network. These type of models are defined by a set of equations based on the mass balance principle. The fluid approach presented by Weiss (1999) incorporates a complete framework, high and low level control, of an entire production facility.

The high level controller, based on a Rolling Horizon Scheme (RHS), uses a multi-class fluid network to approximate the production system. The fluid control problem is a Separated Continuous Linear Program (SCLP) and can be solved numerically, with the use of a simplex based algorithm. Due to this feature, fluid control problems are more tractable than combinatorial optimization problems. An newly developed simplex-based finite algorithm is used to solve the SCLP problem. Similar to the MPC approach, a low level controller translates continues targets into discrete events.

While the above mentioned control framework seems promising, research on fluid models is still in their early stages. Hence the various theorems, models, and results for a full scale re-entrant environment are incomplete.

2.4 Discussion

This chapter surveys several commonly encountered control approaches, both standard and recent developments. The survey illustrates that the majority of research is focussed on single level approaches only. Research on layered control frameworks, combining different approaches at their intended level, is rare. A decomposition of control approaches, based on the decisional aspects, is used to introduce three different control levels; high-level, intermediate-level and low-level.

High level control approaches are typically characterized by their assumed relation for resource capacity. Many commonly implemented 'traditional' approaches, e.g. MRP-I, assume fixed or linear capacity relations. Consequently, they neglect the influence of variability, thus overestimating production capacity and underestimating average flow times.

More recent approaches tend to approximate the characteristic non-linear resource dynamics in various ways. An example of such approach is the iterative simulation proposed by Hung and Leachman (1996). However, convergence is not guaranteed for this approach and is likely to depend on the structure of the underlying production system.

The approach presented by Asmundsson et al. (2003) combines the tractable characteristics of a mathematical programming model with an accurate approximation of resource capacity by deploying approximations from queuing theory. If deployed in an MPC-like structure (Vargas-Villamil and Rivera 2001), adaptability is gained by the parameter estimating layer.

A large variety of different low level control approaches have been introduced , ranging from simple sequencing approaches to extensive control-theoretic approaches. Sequencing rules form the major contributor to the low level control approaches since they guarantee a reasonably good (typically) suboptimal solution in a short time (Pinedo 2001).

Not all sequencing rules work efficiently or result in a stable production system, especially in a complex (re-entrant) production environment. Appropriate sequencing rules are the Earliest Due Date (EDD) and Least Slack (LS) policies. Another promising contribution is the Stochastic Linear Quadratic (SLQ) optimal control theory (Shen and Leachman 2003). Although small-scale experiments show promising results, this approach is still in the early stages of development and can not yet be applied to real-life production systems.

The same holds for the fluid model control approaches (Dai and Weiss 1996, Weiss 1999). They are one of the approaches of the intermediate level control; the grey area between high and low level control approaches. A promising recent contribution to this level of control is the three layer model predictive control framework of Vargas-Villamil and Rivera (2001) and Vargas-Villamil et al. (2003).

Chapter 3

Control framework

As discussed in Chapter 2, most previous research focusses on single level control approaches. Due to the involved computational effort, it is practically impossible to control a full scale re-entrant production environment with these approaches.

The interaction between approaches at different levels is essential in effectively achieving the goals of these production systems. However, research on layered control frameworks combining different approaches at their intended levels is rare. The sole contribution is a three layer Model Predictive Control (MPC) framework, recently presented by Vargas-Villamil et al. (2003).

In this chapter, a two layer framework for Model Predictive Control (MPC) is presented. This framework deploys and integrates different approaches in their intended environment. The general structure and objectives of the control framework are presented in Section 3.1. In the subsequent sections, an in-depth description of the control approaches incorporated in the two layers is provided.

3.1 General framework

Within this thesis, high and low level control are implemented in a hierarchical control framework. Different levels are used to subdivide an otherwise intractable control problem into multiple smaller problems. Although the solution is suboptimal, the computational effort and the frequency of the control decisions are reduced significantly. Control levels are distinguished by both the decisional aspects of the control system (i.e. span, scope and resolution) and the frequency separation of the (controllable) events (Kimemia and Gershwin 1983).

The main objective of the control framework is to translate actual demand into controllable events for a re-entrant production environment. The objective is attained in two steps (layers).

First, the high level controller performs a planning task. This task encompasses 'the allocation of actual demand to available workstation capacity within a period of time, based on the aggregated state of the underlying production system' (planning). The objective is to globally optimize the performance measures. The low level controller is accordingly provided with a feasible set of short–term production targets.

The second step is performed by the low level controller. This step encompasses 'the generation of controllable events (e.g. release of material) based on the set of production targets and on the actual state of the underlying workstation' (direct control). The objective of this level is to locally optimize the production system by striving to achieve the globally determined production targets.

Figure 3.1 reflects the described hierarchical control framework. All control decisions (e.g. equipment acquisition) with a scope beyond the control boundary (dashed line) are assumed to be given.

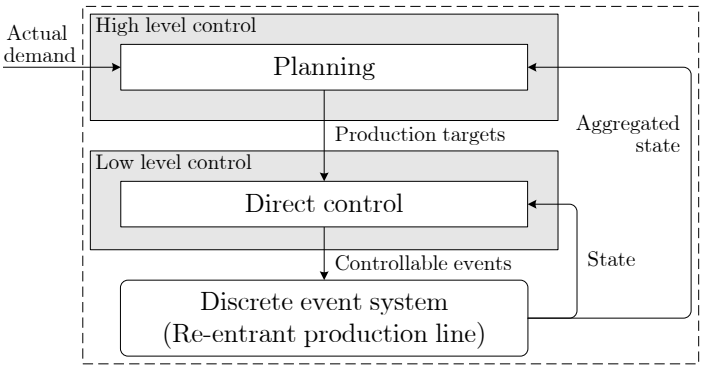


Figure 3.1: Hierarchical two layer control framework

The presented control framework is further explored in the subsequent sections. Section 3.2 and Section 3.3 describe respectively the high level control layer and the low level control layer. Although several approaches have already been introduced in Chapter 2, a more thorough description of the approaches used to attain the objectives of the individual layers is presented. Together with the definitions of the approaches, several commonly encountered issues are discussed, illustrated and resolved. Finally, the complete control framework, encompassing all approaches, is discussed in Section 3.4.

3.2 High level control

As described in the previous section, the high level control layer is used as the primary step in the translation of actual demand into controllable events. Within this layer, planning approaches are used to derive a set of feasible production targets based on the actual demand and the aggregated state of the production system. To accomplish the planning task, high level control approaches commonly deploy an aggregated model of

the production system. The performance of the planning approaches depends heavily on the assumed relation for the resource capacity.

Asmundsson et al. (2003) argue that many planning approaches suffer from a fundamental circularity. Within each planning cycle, the number of lot releases into the production environment is determined based on the flow time estimates (embedded in the resource capacity constraint). The amount of WIP determines the workstation utilization, which in turn influences the realized flow times. In other words, a different (linear) resource capacity constraint (function of estimated flow time) is defined each planning cycle. Eventually, this circularity will converge for steady state environments. However, for dynamic environments convergence is not guaranteed.

The convergence of the resource capacity constraint in a steady state situation is illustrated by Figure 3.2. Here, the development of the linear resource capacity constraint after each planning cycle is shown. Here, the thick black line describes the capacity constraint after convergence.

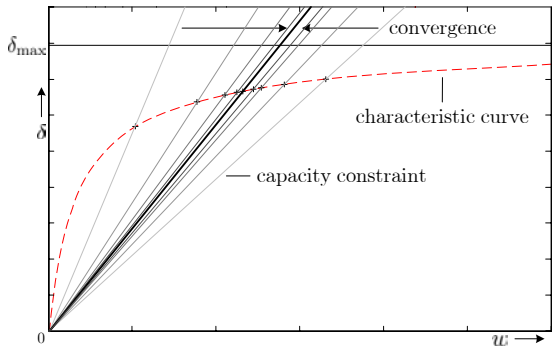


Figure 3.2: Convergence for a steady state environment

Within literature, several approaches exist to circumvent the fundamental circularity. The most commonly encountered approach assumes fixed flow times, independent of system utilization. Many mathematical programming approaches append an additional constraint on the total capacity consumed within a time period. According to this approach, system performance only degrades once the resource capacity constraint is saturated (i.e. 100% utilization). However, both queueing models and industrial experience suggest that system performance should degrade long before reaching 100% utilization.

Another approach to circumvent the fundamental circularity uses iterative Linear Programming (LP) optimization and detailed simulation models (e.g. Hung and Leachman 1996) to approximate the flow times. Iteration continues until a satisfactory agreement between simulation and LP models is obtained. Convergence of such models is not guaranteed and is likely to depend on the structure of the underlying production system. Furthermore the computational tractability decreases rapidly with the size of the underlying production system.

To avoid the problem of fundamental circularity, the characteristic relation between w and δ should be used as a resource capacity constraint by the planning approach. This relation ensures the non-linear degrading of system performance with increasing utilization.

Within this thesis, planning is achieved by a combination of the previously mentioned approaches. Short-term production targets are derived via a Linear Discrete Model (LDM). Within this model, a set of non-linear resource capacity constraints ensures the capacity feasibility. This set of resource capacity constraints is based on clearing functions. The essential parameters required by the clearing functions are estimated using Effective Process Time (EPT) algorithms. Figure 3.3 illustrates the approaches used to accomplish the planning part of the control framework. These approaches are treated in detail in the subsequent subsections.

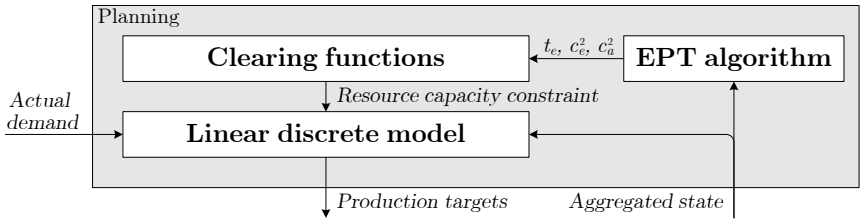


Figure 3.3: High level control layer

Clearing functions

In essence, clearing functions express the expected output as a fraction of the WIP over a given period of time. The term 'clearing functions' was first introduced by Graves (1985) to specify the fraction of the current WIP that can be processed to completion within a period of time. He proposed the use of fixed clearing factors α in a closed queueing analysis (3.1).

$$\delta = \alpha \cdot w \tag{3.1}$$

Srinivasan et al. (1988) and Karmarkar (1989) proposed to use clearing functions as resource capacity constraints in planning models. They extend the idea by replacing the fixed clearing factors α with a non-linear clearing function $\alpha(w)$, described by (3.2). The right hand side of (3.2) is referred to as the clearing function $f(w)$ and is interpreted as a measure of resource capacity.

$$\delta = \alpha(w) \cdot w \tag{3.2}$$

Karmarkar (1989) presents an overview of clearing functions as encountered in literature. A typical example is the 'constant proportion' clearing function, assuming fixed flow time independent of system utilization (recall Figure 2.3).

Secondly, he discusses the 'constant level' function, which introduces a fixed upper bound on the throughput, as deployed in several mathematical programming models. This type of function, without an additional flow time constraint, implies instant production. The third function, the 'combined', is a combination of both previously described functions and is implemented in e.g. mathematical programming model of Hackman and Leachman (1989) and the MRP-C approach (Tardif and Spearman 1997).

The described clearing functions all tend to overestimate the resource capacity. As a solution the 'effective clearing function' is introduced by Asmundsson et al. (2003). This type of clearing function approximates the (non-linear) characteristic curve by deploying approximations from queuing theory.

The advantage of the effective clearing functions is its ability to capture the non-linear resource capacity dynamics while retaining computational tractability. However, Asmundsson et al. (2003) conclude that in general it isn't possible to completely define the effective clearing function due to the myriad of practical details. Instead, Asmundsson et al. (2003) suggest to use an estimate of the effective clearing function based on empirical data; the 'estimated clearing functions'.

Although the estimated clearing functions do capture the practical details, they have one disadvantage. In order to obtain credible results, these estimates can only be used in a steady state environment, e.g. no variation in resource performance.

Within this thesis, essential parameters are estimated using an EPT algorithm (Jacobs et al. 2003). A major advantage of this algorithm is that all sources of variability are captured. This particular topic is discussed later.

The resource capacity can thus be described by approximations from queuing theory, even in a dynamical environment. A commonly encountered approximation for the WIP of a workstation with general inter-arrival and process times, m identical parallel machines and a (parallel processing) batch of size b, is defined by (3.3).

$$w(G/G^k/m) = \frac{b-1}{2} + b \cdot c^2 \cdot \left(\frac{t_0}{mb}\right)^{\gamma} \cdot \frac{\delta^{\gamma}}{1 - \frac{t_0}{mb}\delta} + t_0 \cdot \delta \tag{3.3}$$

$$\text{where } c^2 = \frac{1}{2}(c_a^2/b + c_0^2) \text{ and } \gamma = \sqrt{2(m+1)}$$

In accordance with the definition of the effective clearing function, (3.3) has to be solved for δ (3.4). Here, the non-linear effective clearing function is denoted by f.

$$\delta(G/G^b/m) = f(w, m, b, c) \tag{3.4}$$

Two issues surround the non-linear behavior of (3.3). First of all it is not possible to analytically derive (3.4). Therefore, (3.3) has to be solved numerically. Secondly, (3.4) is non-linear, whereas it should be used in a Linear Programming model. Within this model, the non-linear clearing functions (3.4) are captured by sets of linear approximations (3.5).

In turn, these linear approximations are used as resource capacity constraints.

$$\delta = \boldsymbol{\alpha} \cdot w + \boldsymbol{\beta} \tag{3.5}$$

In the dynamic environment studied here, the clearing functions should be re-determined each period. Because the placement of the linearizations depends on the shape of the underlying clearing function, a linearization-algorithm (described in Appendix A) is embedded within the high level controller. The algorithm returns a set of linearization parameters $\{\boldsymbol{\alpha},\ \boldsymbol{\beta}\}$, describing the individual linear constraints. Figure 3.4 illustrates the approximation of the non-linear characteristic curve of a $G/G/2$-workstation (recall Figure 2.3) by a set of linear approximations.

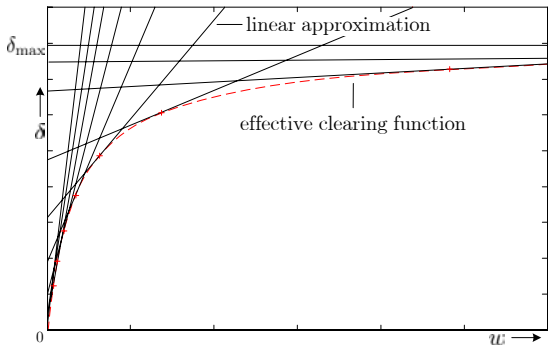


Figure 3.4: Set of linear approximations capturing the effective clearing function

The linearization-algorithm enables the use of effective clearing functions as resource capacity constraints within a linear discrete model. With the use of $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$, the non-linear effective clearing function, f, can be replaced by (3.6).

$$f(w_t) = \min_{\forall c} \left\{ \alpha_t^c \cdot w_t + \beta_t^c \right\} \tag{3.6}$$

Here, the c-superscript represents the individual linear approximations, i.e. $\alpha_t^c \in \boldsymbol{\alpha}_t$. The non-linear clearing function will be described by taking the minimum of one or at most two linearizations. The complete derivation of the effective clearing function from standard approximations from queuing theory to resource capacity constraint is described in Appendix A.

Effective Process Time-algorithm

In order to completely define the effective clearing functions, several parameters have to be estimated each planning cycle. The mean effective process times t_e and the corresponding squared coefficients of variation c_e^2 and the interarrival times c_a^2, are essential parameters (see (3.3)). Although c_a^2 can readily be measured, measurement of t_e and c_e^2 is not as straightforward.

Jacobs et al. (2003) proposed to use the Effective Process Time (EPT) as a performance measure to quantify throughput losses and irregularities in process times. Hopp and Spearman (2000) introduced the effective process time as the time 'seen' by a lot on a server (machine) from a logistical point of view. EPT thus includes setups, down time, repairs, operator unavailability and all other sources of variability.

EPT realizations for a multiple server workstation ($G/G/m$) are obtained by deploying the 'multiple Single Lot Machine' (mSLM)-algorithm (Jacobs et al. 2003, Kock 2003). The EPT realization of lot i on server j of workstation k will be started at $\tau^s_{i,j,k} = \max(\boldsymbol{AA}_{i,j,k}, \boldsymbol{AD}_{i,j,k-1})$. The EPT realization will be ended at $\tau^f_{i,j,k} = \boldsymbol{AD}_{i,j,k}$. With $\boldsymbol{AA}_{i,j,k}$ being the actual arrival and $\boldsymbol{AD}_{i,j,k}$ the actual departure of lot i on server j of workstation k. The EPT realization is represented by (3.7).

$$\boldsymbol{EPT}_{i,j,k} = \boldsymbol{AD}_{i,j,k} - \max\{\boldsymbol{AA}_{i,j,k}, \boldsymbol{AD}_{i,j,k-1}\} \tag{3.7}$$

Distributions are fitted on a set of EPT realizations, using distribution parameters such as the mean effective process time t_e and its squared coefficient of variation c_e^2. Computation of the properties of the EPT distributions are carried out with the use of (3.8). Here, N_k is the total number of EPT realizations on workstation k , Y_n the nth realization and σ_e^2 the variance of the mean EPT t_e.

$$t_e = \frac{1}{N_k} \sum_{n=0}^{N_k-1} Y_n \tag{3.8a}$$

$$\sigma_e^2 = \frac{N_i \sum_{n=0}^{N_k-1} Y_n^2 - \left(\sum_{n=0}^{N_k-1} Y_n\right)^2}{N_k(N_k-1)} \tag{3.8b}$$

$$c_e^2 = \frac{\sigma_e^2}{t_e^2} \tag{3.8c}$$

Application of algorithm mSLM has some implications. Consider the influence that individual EPT realizations have on the distribution properties. In a dynamical environment (fluctuating performance of the equipment), the latest realizations tend to better reflect the state of the system, which is not captured by (3.8a). Accordingly, disturbances in the past have the same influence on the average as recent disturbances. Therefore, a weight factor w_n is introduced. Here, w_n describes the influence of realization n. Figure 3.5 illustrates the weight factor w_n for each realization Y_n with regard to different weight distributions; 'normal', 'moving' or 'exponential weighted moving'.

The 'normal' and 'moving' average regards each realization to be equally important in calculating the average. However, the 'moving average' only regards a limited number of realizations (lookback period). The 'exponential weighted moving' average places more emphasis on the most recent realizations and less on old realizations.

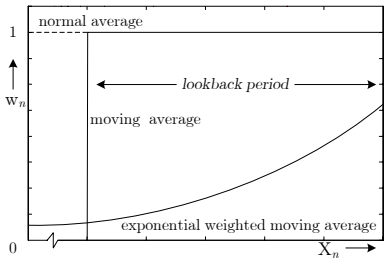


Figure 3.5: Realization weight distributions

The weight factor is implemented by replacing (3.8a) with (3.9). The choice for a weighing distribution depends highly on the dynamical behavior of the underlying production system. Therefore a suitable generic weighing function cannot be selected 'a priori'.

$$t_e = \frac{1}{N_k} \sum_{n=0}^{N_k-1} w_n \cdot Y_n \tag{3.9}$$

Implementation of the EPT algorithm within the control framework enables the use of effective clearing functions as a resource capacity constraint. These resource capacity constraints will form a vital part in the Linear discrete model, which will be discussed in the remainder of this section.

Linear discrete model

In essence, the previous steps provide estimated parameters appertaining to the system state. Based on the system state, the actual planning task of the high level control layer will be performed by a Linear Discrete Model (LDM). Within this model, feasible short-term production targets are derived based on a prediction of the (production) system dynamics (Model Predictive Control). In this thesis, the system dynamics are captured by the effective clearing functions. Since the exact definition of the LDM depends on the underlying production system, an general form will be used to illustrate modeling choices.

An LDM of a production system is a mathematical programming model that describes the state transitions with respect to a set of (capacity) constraints. An LDM is based on two assumptions. First of all, time is discretionalized into fixed periods. In other words, the model describes the relation between model-variables $\mathbf{x}$ (state) of subsequent periods t; $\mathbf{x}_t = f(\mathbf{x}_{t-1})$. Secondly, all relations (describing the state transition) are assumed to be linear.

The general model is formulated as a cost minimization problem (3.10a) subject to a set of constraints. Equation 3.10b reflects a set of equality constraints describing the flow and mass conservation of the production system.

A set of inequality constraints is described by (3.10c). A major contributor to this set are the storage and production capacity constraints. Furthermore, (3.10d) defines a limited range for the model-variables $\mathbf{x}$.

$$\min_{\mathbf{x}} \quad g^T \cdot \mathbf{x} \tag{3.10a}$$

$$\text{s.t.} \quad \mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{3.10b}$$

$$\mathbf{C} \cdot \mathbf{x} \leq \mathbf{d} \tag{3.10c}$$

$$\mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \tag{3.10d}$$

The resource capacity is the most important constraint within an LDM of a production system. In this thesis, the resource capacity constraint is defined as a set of linear approximations of the effective clearing function (3.5).

To implement (3.5), an appropriate measure of WIP of a resource is required. Additionally, the resource capacity needs to be partitioned to be able to address the presence of different products (steps) that consume a resource.

The measure for WIP needs to be redefined since the linear discrete model divides the planning horizon into a number of time periods. The model thus only distinguishes WIP levels at the beginning of a period (i.e. the WIP level at the end of the previous period w_{t-1}) and at the end of a period. Clearly, in reality, the WIP level of a workstation tends to fluctuate during the period, due to the inhomogeneous in- and outflux of the workstation.

A number of alternatives have been proposed to address the issue surrounding the definition of an unambiguous measure of WIP that can be used to represent the situation over the entire period. Srinivasan et al. (1988) propose the use of the WIP at the beginning of the period t, w_{t-1} in a clearing function to establish the capacity during the entire period, i.e. $\delta_t \leq f_t(w_{t-1})$. Karmarkar (1989) suggests to use the WIP at the end of a period t, w_t as a measure of the WIP, i.e. $\delta_t \leq f_t(w_t)$. In both cases, fluctuations in WIP level within a period are omitted. Especially in relatively long periods, these definitions may introduce a significant inaccuracy.

Alternatively, the average WIP level of a period can be used, approximated by the average of the beginning and ending WIP in that period, i.e. $\bar{w}_t = \frac{1}{2}(w_{t-1} + w_t)$. Now, the clearing function is defined as a function of the average WIP during the period, i.e. $\delta_t \leq f_t(\bar{w}_t)$. For this approach, the same average WIP level can be achieved by many different beginning and ending WIP levels, the optimal WIP level tends to oscillate. Preliminary experiments confirm the oscillating behavior as is illustrated by Example 3.1.

> **Example 3.1** Consider a workstation with a constant demand trajectory for product i. To meet the demand, an average WIP level (throughout the period) is required. Figure 3.6 shows the output of the linear discrete model if the resource capacity constraint would use $\bar{w}_t = \frac{1}{2}(w_{t-1} + w_t)$ as measure for the WIP.

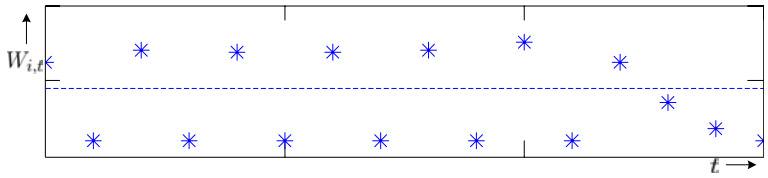


Figure 3.6: Oscillating behavior of the optimal WIP level w_t

The oscillation behavior of w_t around the ideal average WIP level (dashed line) can clearly be seen. ◀

Finally, Asmundsson et al. (2003) propose to reformulate the linear discrete model by offsetting the throughput by half a period from what is traditionally done. Hence δ_t can be used as an estimate of the production rate at time t by dividing it by the length of the time period.

Although the approach of Asmundsson et al. (2003) seems plausible, the alternative presented by Srinivasan et al. (1988), i.e. $\delta_t \leq f_t(w_{t-1})$, is preferred to preserve the formulation of (3.10). Note that this approach restricts the freedom to choose an appropriate period size to avoid significant inaccuracy.

As stated above, the definition of the effective clearing functions involves the ability to partition capacity. In this thesis, the resource capacity constraint for a workstation is defined by a single effective clearing function. However, within a (re-entrant) production system, several products compete for capacity at the same resource.

Note that when an unpartitioned constraint is used, capacity is possibly created for one product by holding WIP for another. Example 3.2 clearly illustrates this problem.

Example 3.2 Consider a workstation producing a single product with a number of re-entrant production steps l. Each step has an equal resource consumption factor. If the resource capacity is un-partitioned, capacity is allocated according to (3.11).

$$\sum_{\forall l} \delta_{l,t} \leq f_t \sum_{\forall l} w_{l,t-1} \tag{3.11}$$

In this case, the optimal solution maintains a high WIP level of the product with least associated cost. The capacity generated by this product can be used to reduce WIP levels of all other products. ◀

A number of alternatives have been proposed to address capacity partitioning. The most commonly encountered alternative is to define a single clearing function for all steps on a workstation and to partition it for each individual step.

Here, the fraction of the total resource capacity consumed by step l is equivalent to its fraction of the total WIP (3.12).

$$\delta_{l,t} \leq \frac{w_{l,t}}{\sum_{\forall l} w_{l,t}} \cdot f_t(w_{l,t}) \tag{3.12}$$

Although (3.12) introduces an adequate resource capacity partitioning approach, it introduces non-linearity, and is thus not applicable to a linear discrete model.

Alternatively, a fixed 'resource capacity consumption factor', describing the capacity consumption for a resource per unit produced, can be introduced (Asmundsson et al. 2003). Since such a factor would be based on historical data, similar to the clearing functions as discussed previously, it would work in a steady state environment, but not in a dynamic environment.

Another alternative would be to define a set of constraints that relates total resource capacity to the total WIP (3.13a), and that relates the resource capacity of step l to the corresponding WIP, w_l, as denoted in (3.13b).

$$\sum_{\forall l} \delta_{l,t} \leq f_t(\sum_{\forall l} w_{l,t}) \tag{3.13a}$$

$$\delta_{l,t} \leq f_t(w_{l,t}) \tag{3.13b}$$

Here, (3.13) is defined for a two step environment. An environment with more than two steps would require a set of constraints for every combination of layers. Therefore an environment with n steps per workstation, would require a set of $2^{n_k} - 1$ capacity constraints for workstation k.

Although the number of constraints increases rapidly with the problem-size, the latter alternative is used in the LDM. Due to the exploding number of constraints, more research towards capacity partitioning is required.

In conclusion, this section introduces a detailed high-level control layer. This control layer performs the first (planning) step by translating the actual demand into production targets and, eventually, into controllable events. Here, fundamental circularity is avoided by deploying the characteristic non-lineair system dynamics as a resource capacity constraint. Furthermore, these constraints ensure the derivation of predictable and feasible short-term production targets which are fed to the low level control layer.

3.3 Low level control

The second step in the hierarchical control framework, direct control, is governed by the low level control layer. This step encompasses the short-term decisions, based on the current state of the production system, in order to achieve the production targets issued by the high level control layer. A survey on commonly encountered low level control approaches is provided in Section 2.2.

Since the low level control layer focusses on local optimization, the low level control layer is distributed over the production system (workstations). Here, a distributed dispatching policy is used since it guarantees a good, yet suboptimal, solution in a short time (Pinedo 2001). The deployed distributed dispatching policy is a combination of two dispatching rules (composite dispatching rule).

First, the issued production targets are used to translate the current buffer content (state) of the workstation into a set of possible work. Secondly, a sequencing policy is used to determine which lot or batch should be dispatched next (dispatched work). Figure 3.7 illustrates the approaches used in the distributed direct control part of the control framework. Both approaches are detailed and discussed hereunder.

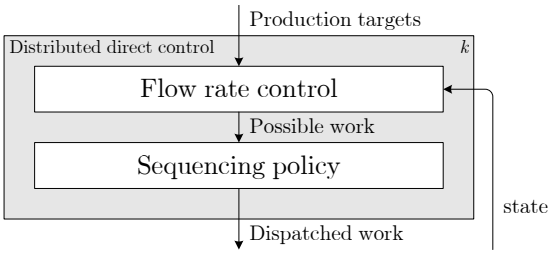


Figure 3.7: Distributed direct control for workstation k

Flow rate control

The targets issued by the high level control layer describe the desired flow of lots through the Discrete Event System (DES). The inflow of the DES is captured by the number of lot releases per unit of time, while lot departures form the outflow of lots. The production targets describe the number of lots that should be processed during period t for each individual workstation.

In essence, the flow rate control rule can be seen as a traffic signal. When (a server of) a workstation becomes idle, the current buffer content is examined. Based on the remainder of the issued production targets, the current buffer content (state) of the workstation is translated into a set of possible work (lots that are allowed to be processed during the current period). If the issued production targets are achieved, the workstation must remain idle. If the workstation has not reached the desired target, a sequencing policy selects the next lot or batch to be processed (dispatched).

Sequencing policy

Many sequencing approaches have been proposed (Blackstone et al. 1982). However, not all sequencing rules perform well in complex (re-entrant) environments. Furthermore, not all sequencing rules are 'stable' (Lu and Kumar 1991) in such an environment.

A sequencing rule is said to be stable if the total flow time of a system does not exceed an upperbound, provided the arrival rate is within the system capacity.

The choice for a sequencing policy depends on the desired optimization goal. In this thesis, a Least Slack (LS) policy is used. The least slack sequencing policies are due date based, with the objective to minimize the variance of the tardiness of all lots. LS policies are proven to be stable in complex environments (Lu and Kumar 1991).

These policies take into account the due dates of the wafer lots and give priority to those lots that have the least amount of slack s_i, i.e., the one that are closest to their due dates or are the most past due. Here, the slack of lot i is defined by (3.14), as the difference between the remaining time before the due date $d_i - \tau$ and an estimate of the remaining flow time ζ_l. Where, ζ_l depends on the production step l of the lot.

$$s_i = d_i - \tau - \zeta_l \tag{3.14}$$

A derivative of the Least Slack policy described above, is the 'Least Slack per remaining process step' (LS/n). Here, slack s_i is defined with respect to the remaining number of process steps (3.15). Various comparative studies show that LS/n performs well in complex re-entrant production systems, and that it reduces both the variance and the mean of the tardiness (Lu and Kumar 1991, Waikar et al. 1995).

$$s_i = \frac{d_i - \tau - \zeta_l}{N - l} \tag{3.15}$$

LS/n (3.15) will be deployed, in the low level control layer, to determine which lot (or batch) to dispatch next. Here, ζ_l will be defined based on either the outcome of the LDM or approximations from queuing theory in combination with the effective process times, t_e. This particular topic will be discussed during the implementation of the control framework.

3.4 Discussion

In this chapter, a two layer hierarchical MPC framework is presented. Hierarchical decomposition is used to subdivide an otherwise intractable control problem into multiple smaller problems. The main objective of the MPC framework is to translate actual demand into controllable events for a re-entrant production environment. This objective is achieved in two steps, a high level control step and a distributed low level control step.

The first step is performed by the high level control layer. Here, a Linear Discrete Model is used to derive production targets based on the aggregated state and the predicted behavior of the production system (Model Predictive Control).

An accurate prediction of the non-linear system dynamics is obtained by implementing resource capacity constraints based on the effective clearing functions (approximations from queuing theory). Essential parameters, required for the definition of the effective clearing functions, are determined by an EPT algorithm.

An advantage of the presented high level control layer is that it circumvents the fundamental circularity present in many other planning (optimization) approaches. Furthermore, essential parameters are based on an objective performance measure, rather than on interpreted historical data.

The second step is performed by the low level control layer. Here, a distributed composite dispatching rule is used to determine which lot (or batch) to process (dispatch) next. First, flow rate control determines the possible work that can be dispatched next, based on the issued production targets. Next, a Least Slack per remaining process step (LS/n) sequencing policy is used to determine which lot or batch should be processed next. The LS/n policy is used because it has been proven to be stable and effective in a complex re-entrant environment.

The described hierarchical MPC framework is reflected by Figure 3.8.

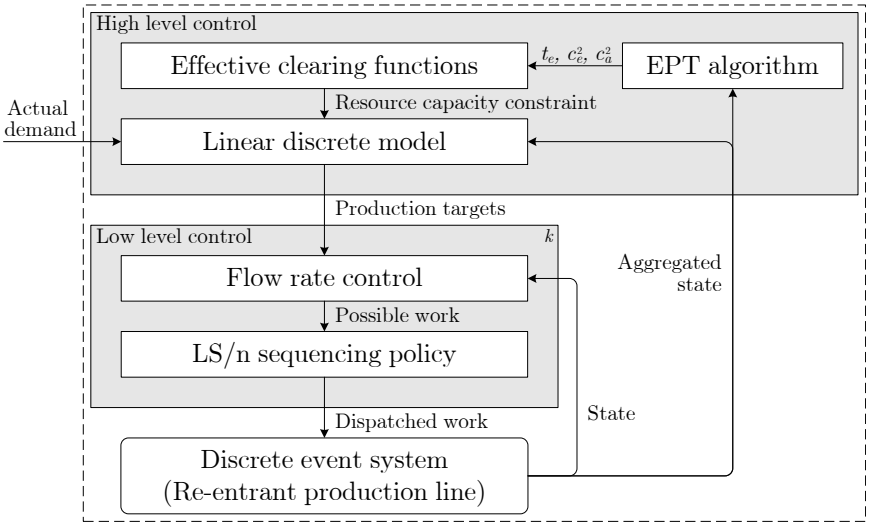


Figure 3.8: Two layer Model Predictive Control framework

The presented control framework resembles the MPC framework as presented by Vargas-Villamil et al. (2003), by distinguishing 'parameter estimation', 'optimization' and 'direct control' sections. However, in contrast to the definition of Vargas-Villamil et al. (2003) the parameter estimation section is considered to be a utility rather than a distinct control layer.

Chapter 4

Cases

In this chapter, two cases are presented, that are used to analyze the performance of the new MPC framework. First, a description of the simulation framework with which the cases and control framework are simulated is presented. In Section 4.2, the 'Intel five-machine six-step Mini-Fab case' and its underlying assumptions is presented. The non-re-entrant case and its corresponding assumptions are discussed in Section 4.3. The re-entrant case is introduced in Section 4.4.

4.1 Simulation framework

Both cases are embedded in a simulation environment deploying several programming languages. The high level control layer is programmed in Matlab 6.1. The discrete event system and low level control layer are programmed in the formalism χ-0.8 (Hofkamp and Rooda 2002). The interaction between both parts is achieved by deploying the Pymat-interface (Sterian 1999) in Python (Lutz and Ascher 1999). The simulation framework is illustrated by Figure 4.1.

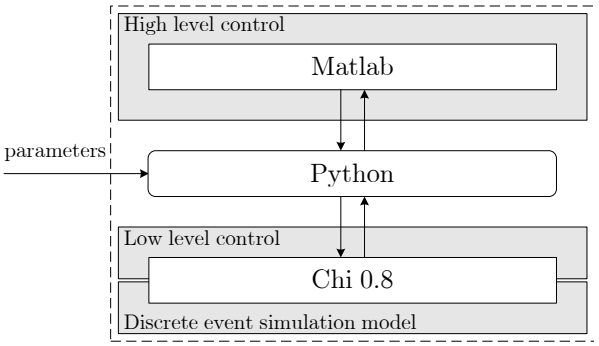


Figure 4.1: Simulation framework

A distinctive part of the simulation framework is formed by high level controller. Conversely, the borderline between the distributed low level control layer and Discrete Event Simulation Model (DESM) is not clear. In fact, the distributed low level control layer is implemented as a set of rules within the DESM. A detailed description of the simulation framework used in this thesis is presented in Appendix B.

4.2 Intel case description

Semiconductor production lines are one of the most complex production environments available. The Intel Case (Kempf 2003) exhibits all characteristic features of such a production line, i.e. reentrancy, different process times, batching, setups, loading and unloading. The case contains six process steps ($l \in \{0, \ldots, 5\}$) and five machines (A, B, C, D, E) which are distributed over three workstations ($k \in \{0, 1, 2\}$). Moreover, it distinguishes two product-flows (P_a, P_b) and one test-flow (TW). All products follow the same predefined route, illustrated by Figure 4.2.

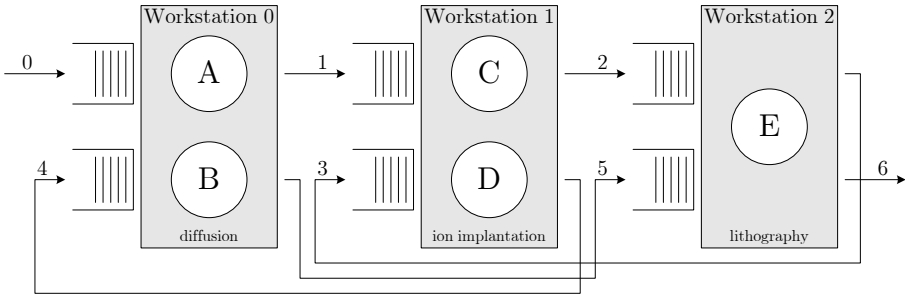


Figure 4.2: Intel Case product flow

The characteristics of the workstations are similar to diffusion ($k = 0$), ion implantation ($k = 1$), and lithography ($k = 2$). Workstation 0 performs steps 0 and 4 and it processes lots in batches of three. Workstation 1 serves steps 1 and 3 and it processes one lot at a time. Workstation 2 serves steps 2 and 5 and requires setup times between step and product change. Individual workstation characteristics are represented in Table 4.1.

k	l	$t_{0,l}$	$t_{h,k}$	b_l	n_k
0	0	225	60	3	2
	4	255	60	3	
1	1	30	30	1	2
	3	50	30	1	
2	2	55	20	1	1
	5	10	20	1	

Table 4.1: Intel Case specification

Here, $t_{0,l}$ is the nominal step process time [min], $t_{h,k}$ is the lot handling (load/unload) time [min], b_l is the batch size [lots] and n_k is the number of machines per workstation. In accordance to many semiconductor production lines, the lithography area (WS2) is the bottleneck of the Intel Case.

In the Intel Case operators and technicians are used for respectively (un)loading and maintenance. Transportation between workstations is handled by a transportation-system capable of transporting a single lot at a time. Several other assumptions underly the Intel Case, i.e.:

- Production occurs in 12 hour shifts; 24 hours per day and 7 days per week.
- Process times are deterministic.
- All resources require preventive maintenance.
- Equipment preemption is not allowed.
- No rework is present.
- The buffers of a workstation have a finite storage capacity, i.e. $s_k = [18, 12, 12]$.

Although the Intel Case is a relatively small scale problem, it features all characteristics commonly encountered in a real semiconductor production facility. Therefore, it is suitable to determine the potential of the control framework presented in the previous chapter.

4.3 Case I: No reentrancy

The first case studies the first three steps of the Intel Case. However, instead of three different product flows, only a single product flow is assumed. The objective of this case is to verify the control framework without resource capacity partitioning.

Case description

Although the first case greatly resembles the Intel Case, several simplifications or alterations have been made, while preserving the characteristic behavior of a semiconductor production environment. A visual representation of the product-flow of the first case is provided by Figure 4.3.

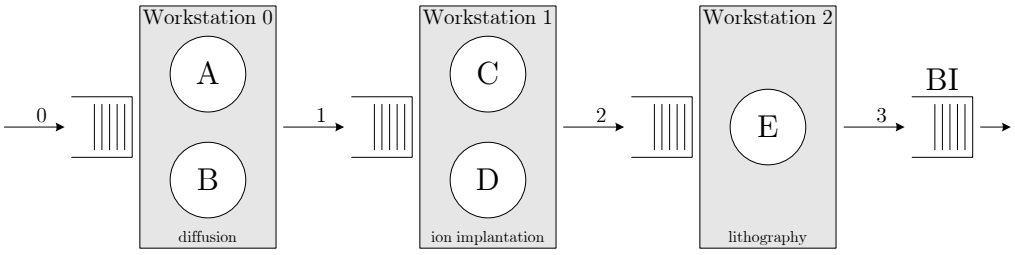


Figure 4.3: Product flow of case I

The following (additional) assumptions underly the first case.

- Single non-re-entrant product flow.
- Sufficient technicians and operators are available.
- Machines are reliable.
- Machines do not require maintenance.
- Transportation time between workstations is negligible.
- After completing the final production step, lots are stored into an inventory buffer BI, where they await customer demand.

The individual workstation characteristics correspond to those of the Intel Case and are represented by Table 4.2.

k	l	t_0	t_h	b_l	n_k
0	0	225.0	60.0	3	2
1	1	30.0	30.0	1	2
2	2	32.5	20.0	1	1

Table 4.2: Workstation characteristics for case I

The following subsections provide an overview of the particular modeling choices concerning the two main components of the simulation framework; the linear discrete model (LDM) and the discrete event simulation model (DESM). The other components of the control framework do not require case-specific choices and are modeled in accordance with their definition discussed in Chapter 3.

Linear discrete model

The actual planning task of the high level control layer is performed by solving the optimization problem. This problem is formulated as a Linear Discrete Model (LDM) of the underlying production system. The general form of this model has already been introduced in Section 3.2.

The linear discrete model of the first case closely follows the formulation of Hackman and Leachman (1989) and Leachman (2001). The underlying production system is modeled as a set of nodes representing the individual production steps.

$$\min \quad \sum_{\forall t} \sum_{\forall l} \Big(\omega_l \cdot W_{l,t} + \iota \cdot I_t + o \cdot BO_t \Big) \tag{4.1}$$

$$\begin{aligned}
\text{s.t.} \quad & W_{l,t} = W_{l,t-1} + R_t - b_l \cdot X_{l,t} && \forall t,\ l = 0 && \text{(4.2a)}\\
& W_{l,t} = W_{l,t-1} + b_{l-1} \cdot X_{l-1,t} - b_l \cdot X_{l,t} && \forall t,\ l = \{1,2\} && \text{(4.2b)}\\
& I_t = I_{t-1} + b_{l-1} \cdot X_{l-1,t} - D_t && \forall t,\ l = 3 && \text{(4.2c)}\\
& BO_t = BO_{t-1} + d_t - D_t && \forall t && \text{(4.2d)}\\
& X_{l,t} \leq \alpha^c_{k,t} \cdot W_{l,t} + \beta^c_{k,t} && \forall c,k,t,\ l \in \mathcal{S}_k && \text{(4.3a)}\\
& W_{l,t} \leq s_k && \forall k,t,\ l \in \mathcal{S}_k && \text{(4.3b)}\\
& X_{l,t},\ W_{l,t},\ D_t,\ I_t,\ R_t,\ BO_t \geq 0, && \forall l,t && \text{(4.4)}
\end{aligned}$$

with
- c : index for linearization parameters
- k : workstation index
- l : step index
- t : time period index
- ι : inventory holding cost
- o : backorder cost
- ω_l : holding cost of step l
- BO_t : backorders at the end of period t
- D_t : actual departures during period t
- I_t : inventory of finished lots at the end of period t
- R_t : actual releases during period t
- $W_{l,t}$: WIP of step l, at the end of period t
- $X_{l,t}$: production quantity of step l, during period t
- b_l : batch size of step l
- d_t : demand for lots, during period t
- s_k : physical storage capacity of workstation k
- $\alpha^c_{k,t}$: c-th gradient parameter for workstation k during period t, i.e. $\alpha^c_{k,t} \in \boldsymbol{\alpha}_{k,t}$
- $\beta^c_{k,t}$: c-th scaling parameter for workstation k during period t, i.e. $\beta^c_{k,t} \in \boldsymbol{\beta}_{k,t}$
- $\mathcal{S}_k$: set of steps belonging to workstation k, i.e. $\mathcal{S}_0 = \{0\}, \mathcal{S}_1 = \{1\}, \mathcal{S}_2 = \{2\}$

As described in the previous chapter, the LDM is formulated as a cost minimization problem (4.1) subject to a set of equality and inequality constraints. Flow and mass conservation of the production system are enforced by the equality constraints (4.2). The 'mass conservation' for step l is defined by (4.2a) and (4.2b). Here, the change in $W_{l,t}$ is defined as the difference between influx and outflux of lots. The flow between production steps is obtained by setting the influx of step l $X_{l,t}$ equal to the outflux of its predecessor $X_{l-1,t}$, illustrated by Figure 4.4.

The two remaining equality constraints, (4.2c) and (4.2d), are used to suppress capacity or demand fluctuations. These two constraints describe respectively the mass conservation of the inventory buffer BI and the backlog of the production system BO_t.

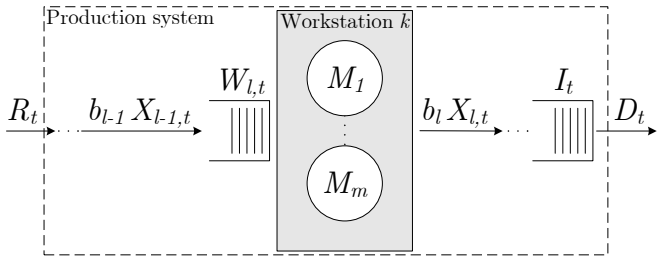


Figure 4.4: Flow and mass conservation within a linear discrete model

Physical capacity limitations are enforced by the inequality constraints (4.3). Here, the physical resource capacity and storage capacity are constrained by respectively (4.3a) and (4.3b). Note that, (4.3a) is in accordance to the definition (3.13b).

Several modeling choices are emphasized briefly:

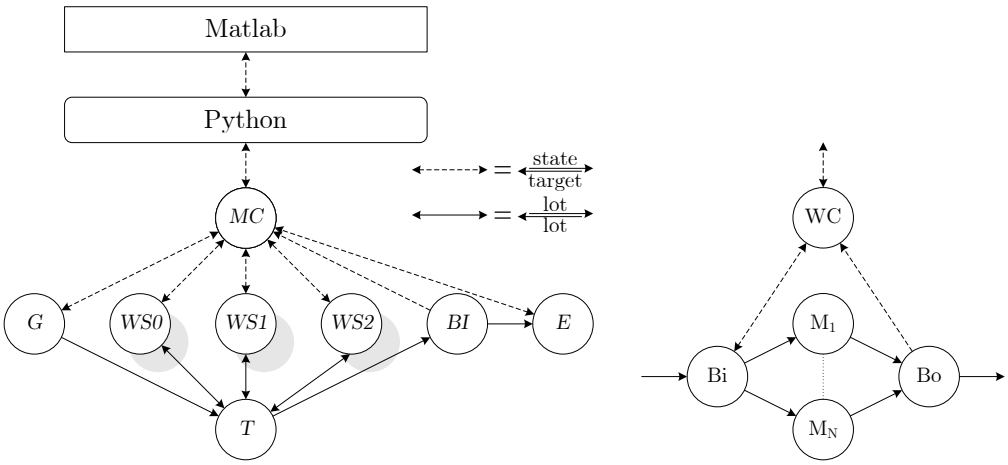
- The allocation of costs (penalties) to variables significantly influences the goal and performance of the optimization problem. Therefore, cost should only be allocated to variables associated with the performance of the optimization problem. In this case, respectively the WIP of step l, $W_{l,t}$, the content of inventory buffer I_t and the number of backorders BO_t.
- Due to the assumed negative correlation between yield and flow time (Chen et al. 1988, Fowler et al. 2002) holding costs ω are related to production progress. Here, production progress is represented by step id l. Due to the (increasing) holding costs, lots are only released (or processed) if necessary.
- At the moment the production system is unable to meet demand d_t a backlog is created. The change in backorders BO_t at the end of period t is defined as the difference between demand d_t and the actual departures D_t (4.2d). A large penalty o is assigned to BO_t to ensure that the LDM will avoid creating a backlog and will prioritize the elimination of an already existing backlog.
- A (finished goods) inventory buffer is placed at the end of the production line to meet demand d_t temporarily exceeding the resource capacity of a single period. Unused resource capacity of earlier periods can be deployed to work in advance to supplement missing capacity. A cost definition consistent with the other buffers (i.e. $\iota > \max\{\omega_l\}$) avoids unintended flow through the production system.
- The inventory buffer provides the ability to meet a demand temporarily exceeding the short-term resource capacity. However, the demand-trajectory d should be attainable over the complete planning horizon. That is, the total demand $\sum_{\forall t} d_t$ should be restricted to the total resource capacity $\sum_{\forall t} X_t \mid_{w=w_{max}}$.

Together with the other components of the high level control layer, the EPT algorithm and the effective clearing functions, the linear discrete model is embedded within a set of Matlab-files. A complete overview of these files and their interaction is presented in Appendix C.

Discrete event simulation model

For the first case the physical production system is replaced by a discrete event simulation model (DESM). As mentioned in Section 4.1, the DESM is programmed in χ-0.8. An advantage of χ-0.8 is that both the material flow as well as the control flow of a production system can be modeled.

Here, the production system is modeled by subdividing it into several processes, based on an organizational decomposition. Lots are released into the production line by the generator (G). A transportation process (T) is used for lot handling between generator (G), workstations (WS) and inventory buffer (BI). The route through the production system is defined by the routing table, mapping production step l to workstation k. For the first case the routing table is defined as $[0, 1, 2, 3]$. Here, step 3 refers to the inventory buffer (BI). In this buffer the lot awaits customer demand and leaves the production line (E). The flow of materials through the production line is illustrated by the continuous arrows in Figure 4.5.



(a) System level (b) Workstation level

Figure 4.5: Material flow and control interaction within the DESM

All workstations are modeled according to the same general form. The workstation is composed of two buffers, a set of identical machines and a workstation controller. A lot enters the workstation through the inbound buffer (Bi). If a machine (M) runs idle, the content of the inbound buffer is evaluated by the embedded low level controller to determine the optimal lot (or batch) to process next. After the machine completes production, the lot is stored in the outbound buffer (Bo), where it awaits transportation to the next process step. The individual workstation characteristics, such as setup times, are defined by a set of parameters. Figure 4.5(b) illustrates the material flow within a workstation.

The original Intel Case assumes a finite storage capacity for the buffers. Within this case, the physical storage capacity is embedded as a constraint within the LDM (4.3b). However, within the DESM the storage capacity is modeled as infinite. Therefore, the content of the buffer can be used as a performance measure of the control framework.

As mentioned earlier, the distributed low level control layer is implemented in the DESM. A hierarchical decomposition is used for the control framework. The heart of the DESM is the centralized controller (MC). This controller triggers Matlab to solve the planning problem based on the state of the production system. Furthermore, it distributes the production targets to the workstations (WS), generator (G) and exit process (E). The control flow within the production system is illustrated by the dashed arrows by Figure 4.5(a).

The production targets are received by the workstation controller (WC). The function of the workstation controller is twofold. First of all it redirects the production targets to the incoming buffer (Bi), at which the actual low level control is embedded. Secondly, triggered by the centralized controller (C), the workstation controller determines the state (i.e. WIP and EPT realizations) of the workstation and sends it upwards. The interaction between different control processes within a workstation is illustrated by Figure 4.5(b). A detailed description of the first case, together with the individual files of the DESM, is provided in Appendix C.

4.4 Case II: Re-entrant product-flow

The first case did not incorporate a re-entrant product flow. Since the applicability of the control framework for a system with a re-entrant product flow still has to be examined, a second case is presented.

Case description

The production system considered in the second case corresponds to the original Intel Case, as illustrated by Figure 4.6.

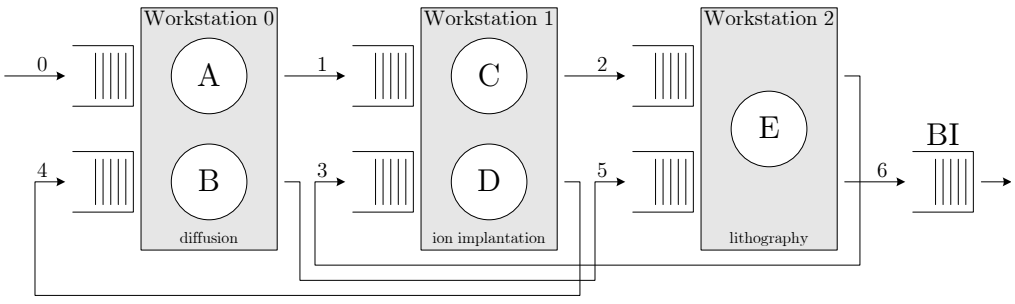


Figure 4.6: Re-entrant product flow of case II

However, in accordance with the first case, the second case only regards a single product. Apart from the product flow, all other assumptions correspond to those underlying the first case. The workstation characteristics are presented in Table 4.3.

k	l	t_0	t_h	b_l	n_k
0	0	240.0	60.0	3	2
	4	240.0	60.0	3	2
1	1	30.0	30.0	1	2
	3	50.0	30.0	1	2
2	2	55.0	20.0	1	1
	5	10.0	20.0	1	1

Table 4.3: Workstation characteristics for case II

Linear discrete model

The Linear Discrete Model (LDM) of the second case closely follows that of the first case. The LDM of the first case is adapted for a re-entrant production system by deploying the capacity partitioning approach defined by (3.13). The LDM of the second case is defined hereunder.

$$\min \quad \sum_{\forall t} \sum_{\forall l} \Big(\omega_l \cdot W_{l,t} + \iota \cdot I_t + o \cdot BO_t \Big) \tag{4.5}$$

$$\text{s.t.} \quad W_{l,t} = W_{l,t-1} + R_t - b_l \cdot X_{l,t} \qquad \forall t,\ l = 0 \tag{4.6a}$$

$$W_{l,t} = W_{l,t-1} + b_{l-1} \cdot X_{l-1,t} - b_l \cdot X_{l,t} \qquad \forall t,\ l = \{1, \dots, 5\} \tag{4.6b}$$

$$I_t = I_{t-1} + b_{l-1} \cdot X_{l-1,t} - D_t \qquad \forall t,\ l = 6 \tag{4.6c}$$

$$BO_t = BO_{t-1} + d_t - D_t \qquad \forall t \tag{4.6d}$$

$$X_{l,t} \leq \alpha^c_{k,t} \cdot W_{l,t} + \beta^c_{k,t} \qquad \forall c, k, t,\ l \in \mathcal{S}_k \tag{4.7a}$$

$$\sum_{l \in \mathcal{S}_k} X_{l,t} \leq \alpha^c_{k,t} \cdot \sum_{l \in \mathcal{S}_k} W_{l,t} + \beta^c_{k,t} \qquad \forall c, k, t \tag{4.7b}$$

$$\sum_{l \in \mathcal{S}_k} W_{l,t} \leq s_k \qquad \forall k, t \tag{4.7c}$$

$$X_{l,t},\ W_{l,t},\ D_t,\ I_t,\ R_t,\ BO_t \geq 0, \qquad \forall l, t \tag{4.8}$$

with $\mathcal{S}_k$: set of steps belonging to workstation k, i.e. $\mathcal{S}_0 = \{0, 4\}$, $\mathcal{S}_1 = \{1, 3\}$, $\mathcal{S}_2 = \{2, 4\}$

The characteristic changes within the LDM with respect to that of the first case are emphasized briefly.

- The single resource capacity constraint is replaced by a set of constraints (4.7a) and (4.7b) in accordance with the definition in Section 3.2. Here, (4.7) not only relates the total capacity to the total WIP, but also the step capacity $X_{l,t}$ to the available WIP $W_{l,t}$.

- The set $\mathcal{S}_k$, defining the allocation of steps l to workstation k, is changed appropriately.

The LDM is embedded within a set of Matlab-files, together with the other components of the high level control layer. A complete overview of these files and their interaction is presented in Appendix D.

Discrete event simulation model

The discrete event simulation model of the second case is identical to that defined for the first case (recall Figure 4.5). The only alteration required is that of the routing table with respect to the new product flow, i.e $[0, 1, 2, 1, 0, 2, 3]$. A detailed description of the second case, together with the individual files of the DESM, is provided in Appendix D.

4.5 Discussion

In this chapter, a simulation framework and two cases are introduced. These cases will be used to evaluate the performance of the hierarchical MPC framework, presented earlier on. Both cases are based on the Intel Case and feature characteristics commonly encountered in a semiconductor production line.

The first case resembles a three step flow line and will be used to evaluate the performance of the control framework without resource capacity partitioning. The second case resembles the original Intel Case and will be used to evaluate the performance in a re-entrant environment (thus requiring resource capacity partitioning).

Chapter 5

Experiments

In this chapter, the performance of the presented MPC framework is evaluated for the two cases described in the previous chapter. In the first section, the setup of the experiments is described and a set of performance measures is defined. Section 5.2 discusses the validation of the individual parts of the control framework. The simulation results of the first case are analyzed in Section 5.3. The second case is discussed in Section 5.4. The chapter is concluded with a discussion on the performance of the control framework.

5.1 Setup of experiments

The performance of the MPC framework is evaluated by conducting several simulation experiment with the two cases, introduced in the previous chapter. Two cases are used to investigate the performance of the MPC framework in both a non re-entrant and in a re-entrant production system. The two cases are implemented in the general simulation framework presented in Section 4.1.

To place the performance of the MPC framework into perspective, the simulation results are compared with the results of an identical production system controlled by the MRP-C approach (Tardif and Spearman 1997). This approach assumes a constant flow time independent of workstation utilization, resulting in a linear relation between production capacity and utilization. In addition, the resource capacity is constrained by an upperbound. Accordingly, the resource capacity constraint of the MRP-C approach corresponds to the first and last linearization of the effective clearing functions.

The presented MPC framework is expected to maintain a higher WIP-level to counteract the effects of variation in interarrival times and process times. Consequently, the variation on the number of backorders is likely to be less for the MPC framework than for the MRP-C approach.

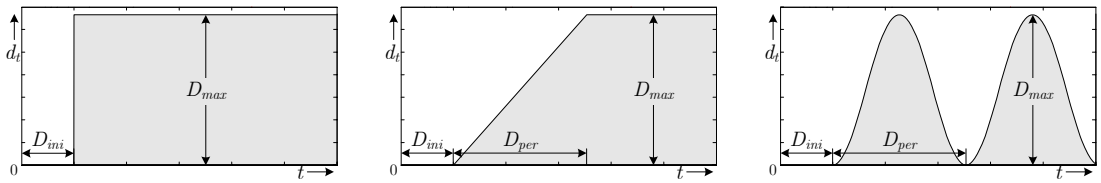
In the remainder of this section, the experiments are specified and the performance measures used to evaluate the control framework are introduced.

Experiments

Three different experiments are used to evaluate the performance of the presented MPC framework. The experiments differ from one another in the predescribed demand trajectory. In the first experiment, an instantaneous change (step) in demand trajectory is enforced. After an initial period D_{ini}, the demand follows a step change from the initial demand D_{min} to its new demand D_{max}, illustrated by Figure 5.1(a). The goal of this experiment is to determine the control framework's ability to look-ahead, work in advance and meet the instantaneous change in demand. Furthermore, the performance in steady state can be evaluated.

In the second experiment, a more subtle change in demand trajectory is enforced. During the transition period D_{per}, the demand is raised linearly to the new steady state value D_{max}, illustrated by Figure 5.1(b). This experiment is used to evaluated the performance during the transient period and the following steady state.

The third and last experiment is used to determine the performance in a continuous transient state. Here, the enforced demand-trajectory describes a sine with period D_{per} and amplitude $\frac{1}{2} \cdot (D_{max} - D_{min})$, illustrated by Figure 5.1(c).



(a) Step up (b) Ramp up (c) Sine

Figure 5.1: Predescribed demand trajectory of the three experiments

The target demand after the transient state D_{max} is set at a bottleneck utilization of around 80%, a value commonly encountered in the semiconductor industry. This bottleneck utilization corresponds to 12 lots per shift for the first case and 6 lots per shift for the second case. In order to check the validity of the observations additional simulation experiments are conducted at lower levels of utilization.

The following additional assumptions underly all experiments:

- Lots released into the production system are available at the beginning of a period. Note that this assumption will induce a high squared coefficient of variation on the inter-arrival time c_a^2.

- Process times are assumed to be stochastic according to a shifted gamma distribution, with mean t_0 and offset $\Delta_0 = 0.8$. The squared coefficient of variation of the total distribution c_0^2 is assumed to be 1.5.
- The resource capacity constraint of the first workstation WS0 is undefined for $w < 1$. Therefore, an initial condition of $w \geq 1$ is enforced.
- Due dates for lots are set as the sum of the effective process times t_e over all process steps, i.e. $d_i = \sum_{\forall l} t_{e,l}$. The estimate of the remaining flow time ζ_l for step l is set appropriately, i.e. $\zeta_l = \sum_l^N t_{e,l}$. Clearly, queuing time at the individual workstations is neglected. Therefore all lots are expected to arrive tardy, however linearly to the due dates set.
- To ensure a fair comparison of simulation results, experiments are conducted with equal process time distributions.
- Several simulation runs with different seeds are conducted to ensure reproducibility of results and observations.

Two steps are performed prior to the execution of the individual experiments. First, the expectations of the outcome of the experiment are defined. Secondly, a brief analysis of the output of the LDM is performed. Both steps are used to assist in the performance evaluation of the MPC framework.

Performance measures

An important performance measure, while evaluating the control framework, is the predictability. Therefore, prior to the simulation experiments the expectations of the outcome of the experiments are mentioned, which are compared to the simulation results. Two questions are examined during the comparison:

- Do the simulation results concur with the expectations?
- If there exists a discrepancy between results and expectations, is it understandable?

In addition to the predictability, additional measures are defined to evaluate the performance of the control framework:

- The total buffer level per workstation with respect to the physical storage capacity of the cases.
- The buffer level per step with respect to the target-levels determined by the LDM.
- The (variation of the) number of exits per period with respect to the enforced demand trajectory.
- The (variation of the) number of backorders (BO) at the end of a period.

These measures are used to enable a performance comparison between the presented MPC framework and the MRP-C approach.

5.2 Validation of individual parts

The performance of the presented MPC framework is investigated by conducting simulation experiments. Before the performance of the MPC framework can be discussed, the different parts of the simulation framework are validated. Three different parts of the defined simulation framework are analyzed; the Dispatching policy, the Discrete Event Simulation Model (DESM) and the Effective Process Time (EPT) algorithm. The correctness of these parts can be validated using three methods (Kleijnen 1995):

- Steady state calculations.
- Visualization of simulation results.
- Manual calculation.

Since the LDM, incorporating the Effective Clearing functions, forms the heart of the control framework, its performance and correctness is reviewed during the experiments.

Dispatching policy

Validation of the dispatching policy is conducted by comparing the output of the dispatching policy with manually derived output. Validation is conducted with respect to various instantiations featuring different characteristics. The validation process of the dispatching policy is illustrated by a small example.

Consider a general workstation at $t = 151$. The process time of the three accompanying steps is defined as 100, 35 and 85. Therefore, the remaining process time from step l, ζ_l, equals $(220, 120, 85)$. Suppose that 10 lots, with different process steps l, release time rt_i and due date d_i, are stored in the corresponding buffer, described by Table 5.1. Here, the latest additions to the buffer is displayed in top of the table.

i	l	rt_i	d_i	s_i
9	1	145	373	0.7
8	1	145	373	0.7
7	1	140	367	-1.3
6	1	135	361	-3.3
5	2	105	330	29.5
4	2	100	324	26.5
3	3	15	238	2.0
2	3	10	232	-4.0
1	3	5	226	-10.0
0	3	0	220	-16.0

Table 5.1: Buffer content

For lot i, the slack per remaining step s_i is defined by (3.15). If the lots are sorted with respect to the least slack per remaining step, the 'optimal' sequence (of lot id's) would be $(0, 1, 2, 6, 7, 8, 9, 3, 4, 5)$. Note that, if lots have equal s_i, lots are sequenced according to the FIFO principle.

If the workstation processes batches of 3 lots and the remaining target $X_{l,t}$ equals $(2, 0, 2)$, the optimal batch sequence (with respect to production targets), would equal $(0, 1, 6), (7, 8, 9)$. Since the remaining production targets do not allow a third lot with step id 3 to be processed, lot 2 is ignored. Therefore, the optimal batch to process next would be composed of lot id 0, 1 and 6. Both sequencing policy and manual calculations derive identical sequences.

Discrete Event Simulation Model

The DESM is validated by comparing the output of a simulation experiment to manual calculations. To obtain a better insight, the state of the system is visualized using Gantt charts. Possible blocking or incorrect implementation of control strategies can be noticed easily.

To rule out any influence of the high level control layer, validation experiments are conducted using fixed (production-)targets. Furthermore, deterministic process times ensure the reproducibility of the simulation results. Some examples of the Gantt charts are presented in Appendix E.1.

The validation of the DESM, by comparing the output of a simulation experiment to manual calculations, resulted in two identical Gantt charts. Therefore, the DESM satisfies the expectations and behaves correctly with respect to the assumptions made.

Effective Process Time algorithm

Validation of the EPT algorithm is performed using two methods; visualization and steady state calculation. Similar to the validation of the DESM, the output (realizations) of the EPT algorithm can be visualized via a Gantt chart. On the basis of buffer-level information and a production Gantt chart, it is possible to manually derive an EPT realization Gantt chart. This manually derived EPT Gantt chart can be compared to a Gantt chart derived from simulation results.

A comparison between simulation output and manually derived EPT Gantt chart indicated that the (controlled) system under observation is subject to 'condition blocking' (Weber 2003). Condition blocking encompasses all types of blocking that occur since a certain condition on the resources is not (yet) met. The controlled production system features condition blocking at the moment production is not allowed (condition $X = 0$) while lots are available for processing.

Since condition blocking is a consequence of a control action, it should not be part of an EPT realization. However, the implemented EPT algorithm (3.7) does not distinguish any form of blocking. Consequently, the condition blocking time is allocated to an EPT realization, thus corrupting the data. The occurrence of condition blocking is illustrated in Figure 5.2. In this figure, the state of a workstation is expressed by the buffer content per workstation and a production Gantt chart. At the moment production targets are met ($X = 0$), production is halted until new targets are received ($X > 0$). The EPT realization $\boldsymbol{EPT}_k$ of lot k (after receiving a new target) is defined (3.7) as the difference between $\boldsymbol{AD}_k$ and $\boldsymbol{AD}_{k-1}$. Consequently, the condition blocking time is allocated to the EPT realization, illustrated by the upper EPT Gantt chart.

The preferred situation is described by the lower EPT Gantt chart. Here, condition blocking is correctly ignored by the EPT algorithm.

However, EPT algorithms for production systems subject to (other forms of) blocking (Kock 2003) are still under development and not yet applicable.

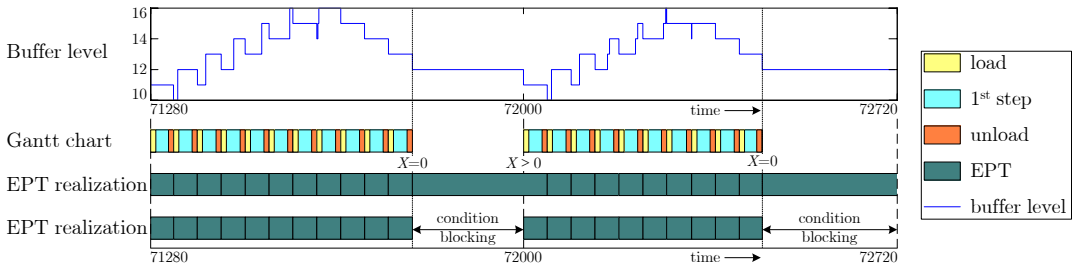


Figure 5.2: Condition blocking with the mSLM EPT algorithm

A suggestion to circumvent condition blocking, would be the introduction of a new parameter, $\boldsymbol{OR}_{i,j,k}$, defined as the moment of order-release of lot i on server j of workstation k. The start time of the EPT realization, $\tau^s_{i,k,l}$ would then be defined as the maximum of $\boldsymbol{AA}_{i,j,k}$, $\boldsymbol{AD}_{i,j,k-1}$ and $\boldsymbol{OR}_{i,j,k}$. Consequently, EPT realization $\boldsymbol{EPT}_{i,j,k}$ would be redefined by (5.1).

$$\boldsymbol{EPT}_{i,j,k} \quad = \quad \boldsymbol{AD}_{i,j,k} - \max\{\boldsymbol{AA}_{i,j,k}, \boldsymbol{AD}_{i,j,k-1}, \boldsymbol{OR}_{i,j,k}\} \tag{5.1}$$

Further research would be necessary to determine the correctness of (5.1).

Another observation during the analysis of the output of the EPT algorithm, is a startup effect. Due to the absence of a history, EPT parameters (t_e, c_e^2 and c_a^2) tend to fluctuate before converging to a stable value. Consequently, online measurement of the EPT parameters, without a history, would have a significant unwanted influence on the resource capacity constraints. To circumvent the startup effect, the simulation experiments will be conducted with fixed estimates for the EPT parameters. If possible, analytical derived parameters will be used.

EPT quantification experiments, with a run length of around 100.000 lots, are used to determine the EPT parameters for both cases. To avoid condition blocking a push approach is used combined with infinite production targets.

These quantification experiments enable a second method of validation; comparing simulation results with analytical steady-state calculations. By deploying standard approaches, it is possible to analytically derive the mean effective process time t_e, and its squared coefficient of variation c_e^2. For the first case, it is also possible to derive the squared coefficient of variation of the inter-arrival time c_a^2.

For stable systems, the squared coefficient of variation of the inter-arrival time c_a^2 equals the squared coefficient of variation on the inter-departure times c_d^2 of its upstream neighbor. A reasonable approximation for the coefficient of variation on the inter-departure times is the linking equation (Hopp and Spearman 2000).

$$c_d^2 \quad = \quad 1 + (1 - u^2) \cdot (c_a^2 - 1) + \frac{u^2}{\sqrt{m}} \cdot (c_e^2 - 1) \tag{5.2}$$

The results of both analytical calculation and simulation experiments are summarized in Table 5.2 and Table 5.3 for respectively the first and second case. Comparison between simulation results and analytical derived parameters yields the observation that both values correspond, implicitly validating both the DESM and EPT algorithm.

			analytical			simulation		
	t_0	t_h	t_e	c_e^2	c_a^2	t_e	c_e^2	c_a^2
workstation 0	225.0	60.0	285.0	0.93	11.0	285.76	0.93	11.00
workstation 1	30.0	30.0	60.0	0.38	4.56	60.05	0.38	4.54
workstation 2	32.5	20.0	52.5	0.57	2.00	52.47	0.56	2.17

Table 5.2: EPT validation results for Case I

				analytical		simulation		
	$t_{0,1^{st}}$	$t_{0,2^{nd}}$	t_h	t_e	c_e^2	t_e	c_e^2	c_a^2
workstation 0	240.0	240.0	60.0	300.0	0.92	300.39	0.88	2.46
workstation 1	30.0	50.0	30.0	70.0	0.52	69.02	0.56	1.58
workstation 2	55.0	10.0	20.0	55.0	0.77	54.97	0.87	1.75

Table 5.3: EPT validation results for Case II

Now the individual parts of the control framework are validated, it is possible to evaluate the performance of the control framework based on the results from the simulation experiments. The results of the simulation experiments, described in Section 5.1, are presented in the following two sections.

5.3 Case I

In this section, a summary of the results and observations obtained from the experiments described in Section 5.1 is presented. A complete overview of the simulation results of the first case can be found in Appendix E.2.

Experiment 1: Step up

Within the first experiment, an instantaneous change (step up) in demand is enforced. Prior to the actual execution of the first experiment, the output of the LDM is analyzed.

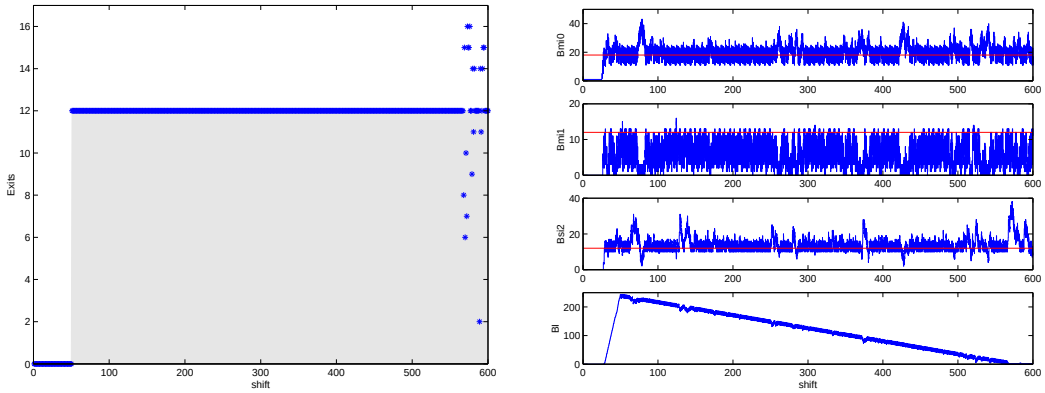
The LDM model converges to an optimal solution for almost all instantiations. However, a single instantiation exists for which the model does not converge. This instantiation is defined by two conditions. If no WIP is present at WS1 (condition 1) and production is likely to start in the first period of the planning horizon (condition 2), the Matlab solver will not converge to an optimal solution.

Consequently, it will reach the maximum number of iterations and provide a non-optimal solution. There is however no clear physical interpretation for such behavior.

Preliminary experiments indicate that this observation occurs with all experiments described in Section 5.1. This signifies either a flaw in the implementation of the LDM or that the optimization problem is too complex for the optimization toolbox of Matlab. However, due to a lack of time this convergence issue has not been resolved within this thesis.

Since condition 1 and 2 can become true in normal operation, iterative use of the control framework does not guarantee an optimal solution, especially with stochastic process times. The output of the LDM, obtained for instantiations where these two conditions are not met, seem to provide correct production targets. Therefore, a rolling horizon scheme is implemented for all experiments conducted with the first case. The resulting production targets are fed to the production system in the corresponding periods. To cope with the effects of variation and real valued targets, newly issued production targets are increased by the residue of prior targets.

Figure 5.3 shows the characteristic results of the first experiment. Here, Figure 5.3(a) displays the number of exits per period with respect to the enforced demand trajectory (grey area). Figure 5.3(b) displays the buffer level for all workstations, with respect to the maximum storage capacity.



(a) Number of exits (b) Buffer level

Figure 5.3: Typical simulation results for the ‘step up’ experiment

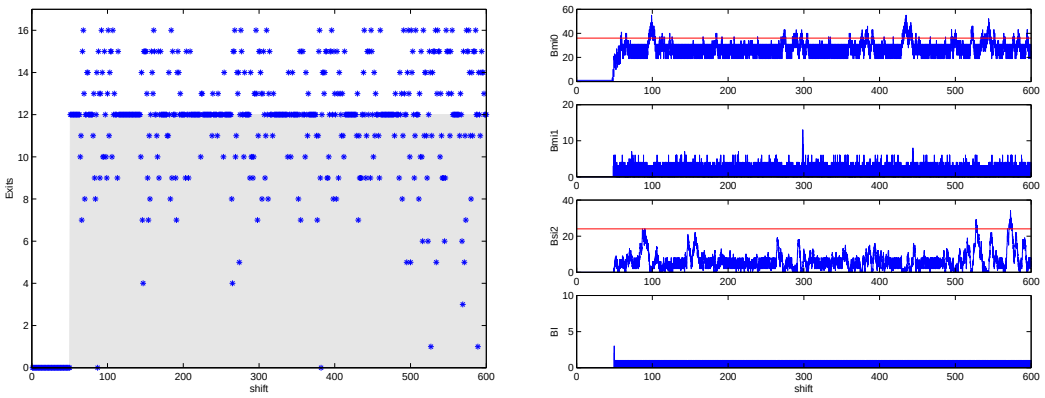
Figure 5.3(a) clearly illustrates that, for the majority of the simulation runs, the number of exits per period does not exhibit any variability. In other words all demand is met on time. However, this observation does not correspond to the expectations, which can be explained as follows.

Due to the high c_a^2 of the first workstation, the LDM aims to maintain a high WIP level corresponding to the resource capacity constraint.

However, the number of lots present within the buffer of WS0 is restricted by the maximum storage capacity (18). In this case, the desired WIP level cannot be obtained. Therefore, the maximum resource capacity is constrained by the maximum storage capacity and the required production quantity (4 batches per shift) cannot be met. To overcome this problem (avoid backorders), the LDM will initiate production before demand requires, to buffer the missing production capacity. Due to the holding cost, all buffers prior to the 'unconstrained' inventory buffer BI will be filled first, as illustrated by Figure 5.3(b). Here, the maximum storage capacity is illustrated by the horizontal line.

Although the number of exits per period exhibits no variability, extremely high queuing times are induced. Since there is a negative correlation between flow time and yield loss, high flow times form a negative effect on the performance of the production system.

The storage capacity constraint of the Intel Case is defined for a deterministic typesetting. For the remainder of the experiments of the first case, the storage capacity of the individual buffers is doubled, i.e. $s_k = [36, 24, 24]$.



(a) Number of exits

(b) Buffer level

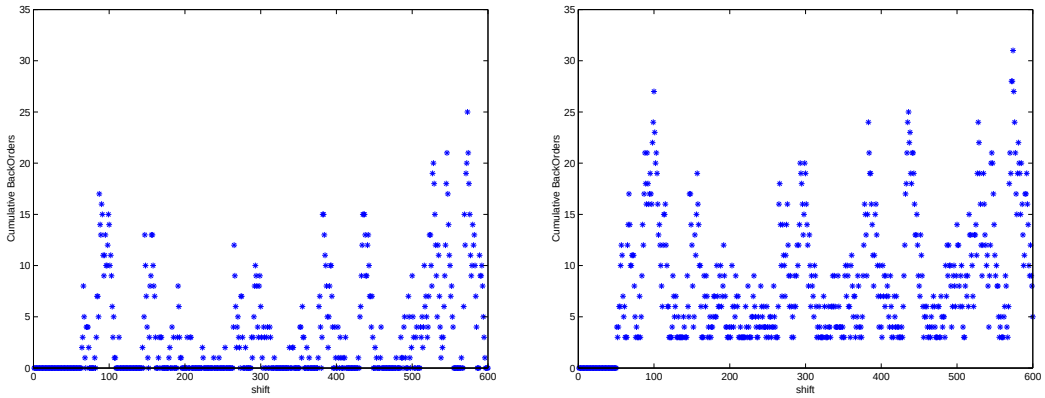
Figure 5.4: Typical simulation results for the 'step up' experiment with altered s_k

Figure 5.4 shows the typical results of the same experiment with the new storage capacity constraint. By doubling the storage capacity, the behavior shown in Figure 5.3 is avoided.

Based on the simulation results, a number of additional observations can be made.

- The system will work in advance to successfully meet the first demand.
- The number of exits will reach a steady state with some variation.
- The mean buffer levels are in correspondence with the output of the LDM.
- The mean measured total flow time is 15% higher than the analytical derived total flow time, potentially caused by a too high security WIP level.

Figure 5.5(a) describes the cumulative number of backorders per period during the first experiment. The results of the same experiment with the MRP-C approach are shown in Figure 5.5(b). As can be seen, the variation on exits, and thus on the number of backorders, is less with the MPC framework than with MRP-C. Furthermore, Figure 5.5(b) reveals that MRP-C is unable to meet the (entire) demand of the first period. Therefore a steady state situation is obtained with a small, but persisting, backlog.



(a) Cumulative BO for MPC (b) Cumulative BO for MRP-C

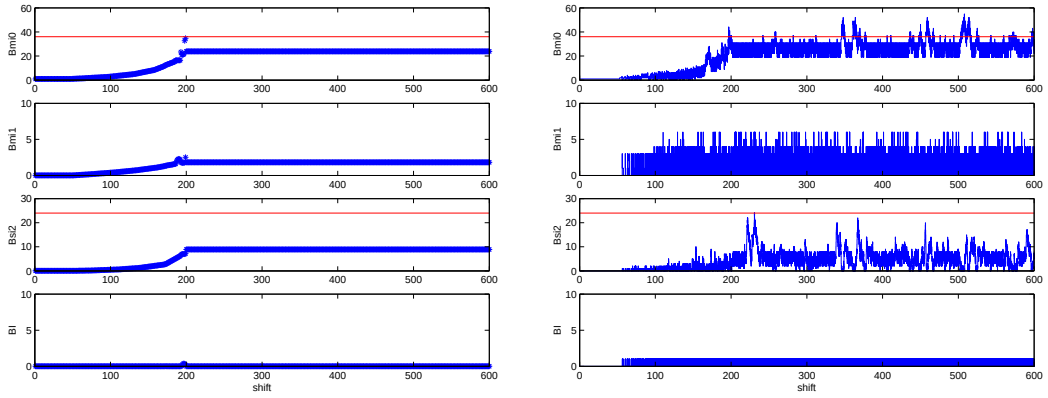
Figure 5.5: Comparison between MPC and MRP-C on basis of cumulative backorders

Experiment 2: Ramp up

In the second experiment, a ramp up in demand is enforced. The observations made based on the results of the ramp up experiment coincide with the observations of the previous experiment. A number of additional observations can be made related to the transient state of the ramp up experiment.

Figure 5.6 shows the buffer level during the ramp up experiment, based on respectively the prediction of the LDM and the actual simulation. Clearly visible is the non-linear increase in WIP, enforced by the non-linear resource capacity constraint, during the linear transient state. A closer look at Figure 5.6(a) even reveals the influence of the individual linear approximations of the effective clearing function.

Figure 5.6(a) furthermore shows a 'bullwhip-effect' in the predicted buffer levels. This effect can be explained as follows. If the production at the last step (WS2) reaches its maximum, additional WIP is needed, which is requested from its preceding step (step 1 at WS1), resulting in a peak load in the corresponding production targets. This will result in two occasions of maximum capacity. Consequently, WS1 requests additional WIP from its predecessor. This phenomena spreads through all production steps. Therefore, WS0 exhibits two additional peak loads in both production and buffer content.



(a) Buffer level predicted by LDM (b) Buffer level according to simulation

Figure 5.6: Buffer levels during the 'ramp up' experiment

The cumulative number of backorders shows a small, but persisting, variation during the transient period. This behavior can be explained by the fact that both demand-trajectory and (production) targets are real valued. The production system is however only capable of producing discrete number of products. Therefore, the dispatching policy will only release a job if the target allows production of a whole lot ($X \geq 1$). If this condition is not met, the system will wait until a new target is received.

Implicitly, within the current control framework the targets are rounded towards minus infinity (floored), creating a small backlog. A number of other strategies for target handling could however be implemented. For instance rounding the targets towards the nearest integer (round), or towards infinity (ceil). All solutions to the discrepancy between discrete production and real valued production targets, are however suboptimal. Flooring a targets creates a small backlog, whereas ceiling a target creates an unwanted product flow. Even rounding a targets does not guarantee an optimal solution. Another solution would be the use of a (mixed-) integer optimization problem, possible a subject for further research.

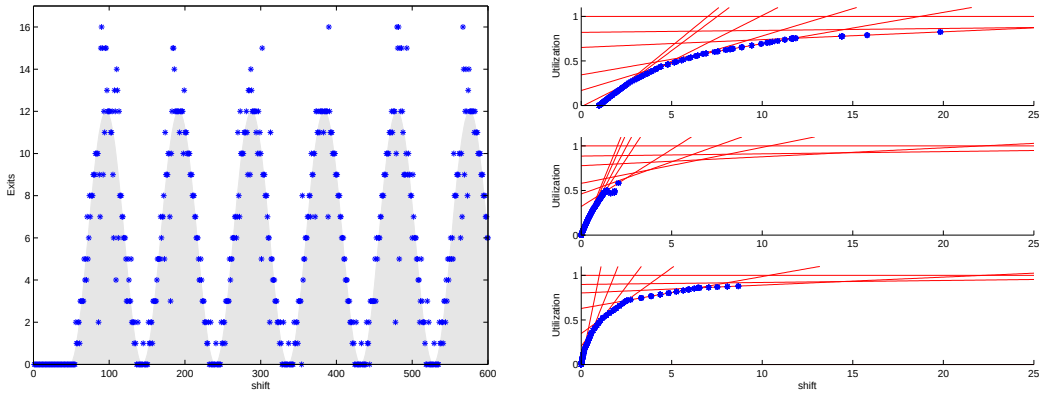
In correspondence to the step up experiment, the influence of variation on the performance of the production system is less with the MPC framework than with MRP-C. The production system controlled by the MRP-C approach shows a smal, but persisting backlog, caused by an overestimation of the production capacity. Therefore, the approach is unable to meet the (entire) demand of the first period.

Experiment 3: Sine

As mentioned during the setup of experiments, the third experiment is used to evaluate the performance of the control framework in a continuous transient state.

The observations for the third experiment are expected to be similar to those of experiments 1 and 2. Evaluation of the results of the experiment confirm these expectations. A number of observations, in particular concerning the transient state, will be described briefly.

Figure 5.7 shows the results of a typical simulation run. The output of the production system is illustrated by Figure 5.7(a). Clearly visible is the discrete output, with respect to a real valued demand trajectory (grey area). The discrepancy between discrete output and real valued demand creates a small persisting variation in the cumulative backorders, illustrated by Figure 5.8. This figure reveals that the small persisting variation in the cumulative backorders is present within both control systems.

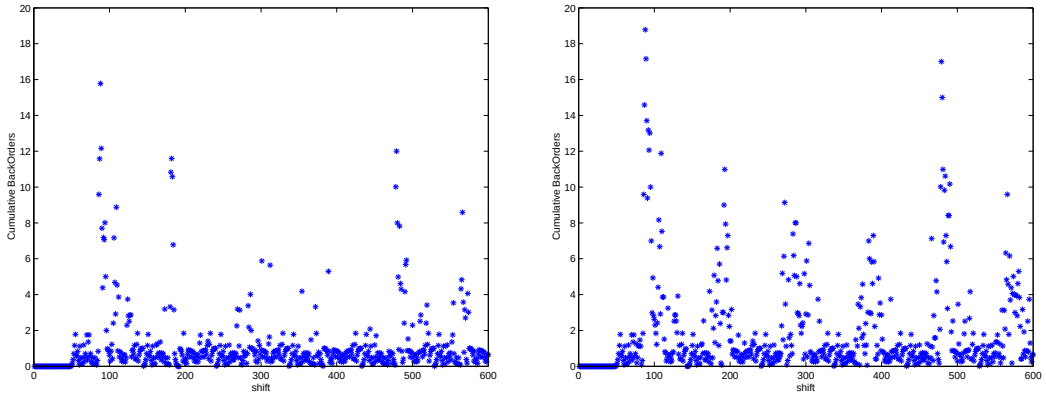


(a) Number of exits (b) Production targets vs. resource constraints

Figure 5.7: Typical simulation results of the 'sine' experiment

The difference between the MPC framework and the MRP-C approach is illustrated by comparing the cumulative number of backorders of the two control approaches. Note that the difference increases with increasing demand (utilization). This difference can be explained by the fact that at a low utilization levels both resource capacity constraints are, by approximation, identical. Recall that the resource capacity constraint of MRP-C is assumed to be equal to the first and last linear approximation of the effective clearing function, illustrated by Figure 5.7(b).

At high utilization levels, the difference between the two resource capacity constraints is extensive due to the fact that MRP-C does not take into account the non-linear behavior the system is likely to follow. Therefore, MRP-C will maintain lower (security) WIP levels than required to counteract the observed variability. Consequently, the variability in process-times will have more influence on the output of the production system, resulting in a higher variability in the cumulative number of backorders, as illustrated by Figure 5.8(b).



(a) Cumulative BO for MPC (b) Cumulative BO for MRP-C

Figure 5.8: Comparison between MPC and MRP-C on basis of cumulative backorders

Implications

A large number of different observations have been made while evaluating the experiments of the first case. For a single instantiation, the LDM will not converge to an optimal solution. This implicates either a flaw in the implementation of the LDM or that the problem is too complex for the optimization toolbox of Matlab. By applying a rolling horizon scheme, an optimal solution can be obtained and performance of the MPC framework can be evaluated. The presented MPC framework reduces the variability on the number of exits per period by maintaining a higher WIP level. The difference between the presented MPC framework and MRP-C increases with increasing utilization. Due to a discrepancy between the real valued production targets and the discrete production system, a small persisting variation is present in the output of the production system.

The majority of observations for the first case are expected to be case independent and can be observed in the simulation results of the second case.

5.4 Case II

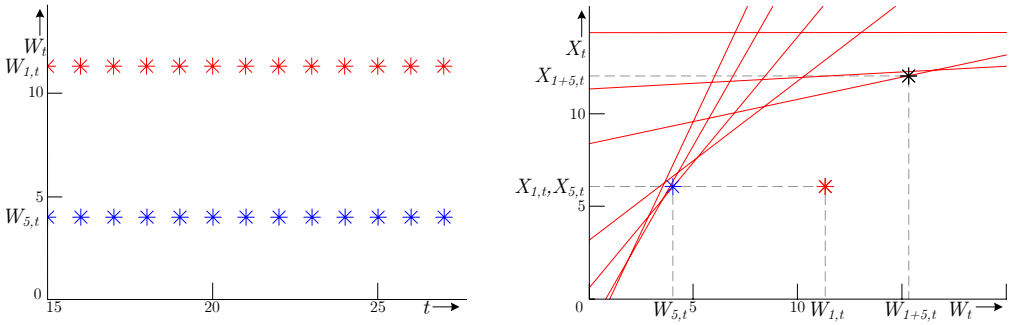
The second case is used to evaluate the performance of the MPC framework with regard to a re-entrant production system. The majority of observations made for the first case are expected to apply to the second case. Since utilization levels of the two cases correspond, all experiments are conducted with a double sized storage capacity, i.e. $s_k = [36, 24, 24]$. A complete overview of the simulation results of the second case can be found in Appendix E.3.

Experiment 1: Step up

Prior to the actual execution of the first experiment, the output of the LDM model, especially the performance of the resource capacity partitioning approach (described in Section 3.2), is analyzed.

Figure 5.9 illustrates the output of the LDM for the first workstation WS0, serving steps 1 and 5. Figure 5.9(a) displays the WIP targets in steady state after the step change. In contrast to the expectations, the required WIP is not partitioned equally over the two steps. However, the two production steps meet the individual capacity requirements defined by (3.13b). Therefore, the unwanted behavior that capacity is created without WIP present (recall Example 3.2), is avoided.

The discrepancy between both WIP levels is procured by the allocation of the WIP additionally required to meet the total capacity (3.13a). In order to meet the total capacity ($X_{1+5,t}$) additional WIP is required in addition to the sum of the WIP required by the individual steps. In other words, the sum of the whole ($W_{1+5,t}$) is greater than the sum of its parts ($W_{1,t} + W_{5,t}$). This is the result of the non-linear relation between capacity and WIP, as illustrated by Figure 5.9(b).



(a) WIP reference trajectory (b) Resource capacity constraint

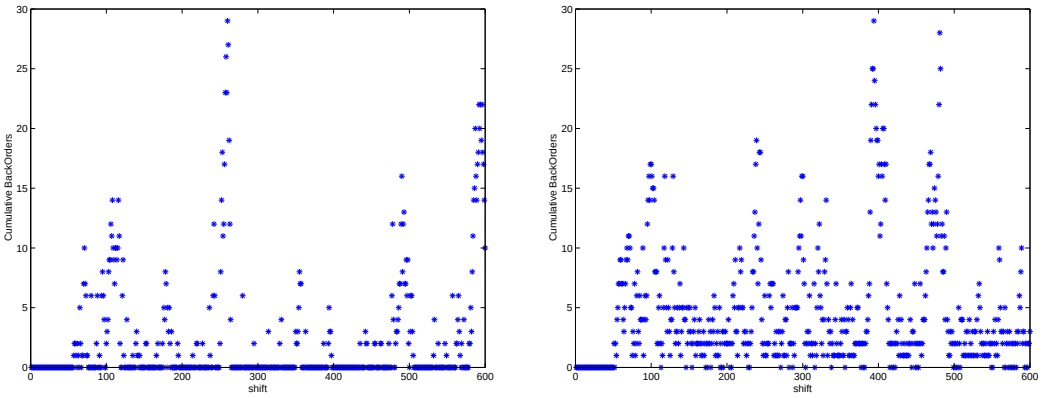
Figure 5.9: Resource capacity partitioning in steady state

The required additional WIP is allocated by the LDM with respect to the cost definition. Due to the linear increasing holding cost, additional WIP is allocated to the first step of the workstation. Note that due to the linear resource capacity constraint used by MRP-C, WIP is distributed equally over the production steps.

The implication of an unequal WIP distribution, is expected to be somewhat contradictory. Due to the unequal distribution of WIP, process time variability c_e^2 will have a larger influence on the (variation of the) output of the production system. On the other hand, the unequal WIP distribution decreases the effects caused by the variability in interarrival times c_a^2. Since, for these simulation cases, EPT experiments indicate that for all workstations $c_a^2 > c_e^2$, the effect of the unequal WIP distribution is expected to be positive with regard to the (variation of the) output of the production system.

Further analysis of the LDM reveals a single instantiation, defined by two conditions, for which the model does not converge. If no WIP of the first step (step 1) is present at WS1 (first condition) and production is likely to start in the first period of the planning horizon (second condition) the LDM will not obtain an optimal solution. This observation is consistent with a similar observation of the first case and is likely to have the same cause. A rolling horizon scheme is used to evaluate the performance of the MPC framework.

The results of the first experiment are completely in correspondence to the expectations. WIP is distributed unequally over the first and second step of a workstation, therefore reducing the effects of c_a^2. Figure 5.10 illustrates the number of backorders of the production system controlled by both control approaches. By maintaining a higher security WIP level, the MPC framework is able to reduce the effects of variability, illustrated by Figure 5.10(a). The production system controlled by the MRP-C approach, shows a higher degree of variability in the number of backorders, illustrated by Figure 5.10(b).



(a) Cumulative BO for MPC

(b) Cumulative BO for MRP-C

Figure 5.10: Typical simulation results for the 'step up' experiment

Experiment 2: Ramp up

Within the second experiment the second case is subjected to a ramp up in demand. The observations, made based on the results of this experiment, are in accordance with the expectations and observations made with pervious experiments. Several observations are summarized briefly.

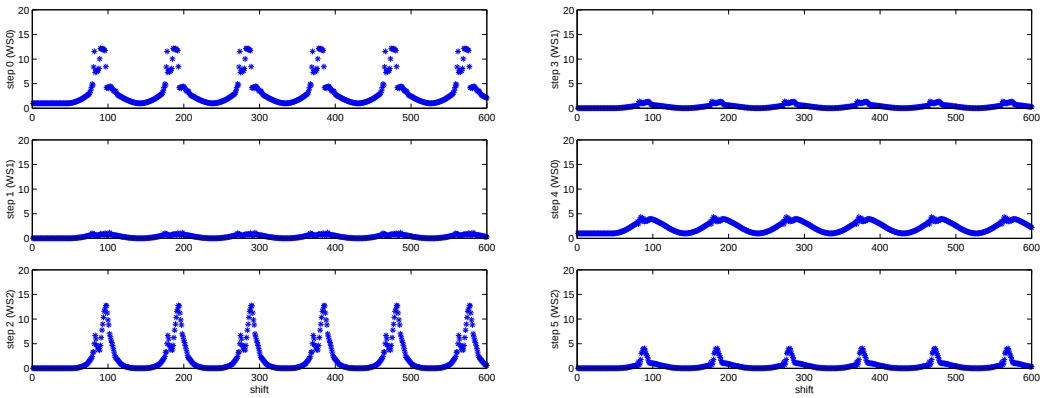
- During the transient state a small, but persistent variation in the cumulative backorders is caused by the real valued targets and demand-trajectory.
- The (predescribed) buffer level clearly shows a non-linear increase with a linear increase in demand (utilization), caused by the non-linear resource capacity relation.

- The bullwhip-effect observed at previous experiments can also be found within the results of this case.
- The variation in the number of backorders of the production system is less for the system controlled by the MPC framework than for the system controlled by the MRP-C approach.

An overview of the results of the second experiment can be found in Appendix E.3.

Experiment 3: Sine

Analysis of the results of the third experiment do not yield additional observations with respect to previously discussed experiments. The results do however clearly illustrate the prolongation of the bullwhip-effect through the production system. This effect is illustrated by Figure 5.11, describing the WIP-level per step predicted by the LDM.



(a) Predicted WIP levels step 0 to 2

(b) Predicted WIP levels step 3 to 5

Figure 5.11: WIP levels per production step predicted by the LDM

An overview of the results of the third experiment can be found in Appendix E.3.

Implications

The majority of observations for the second case correspond to those made for the first case. For a single instantiation, the LDM will not converge to an optimal solution. This indicates either a flaw in the implementation of the LDM or that the problem is too complex for the optimization toolbox of Matlab. Additional WIP required to meet the total capacity is allocated with respect to the holding cost. The bullwhip-effect observed within both cases is clearly illustrated within the results of the second case.

The MPC framework seems to outperform the MRP-C approach with respect to the (variation in the) output of the system.

5.5 Discussion

In this chapter, two cases have been considered for evaluation of the performance of the presented MPC framework. The control framework is evaluated in two steps. First, the individual parts of the control framework were validated. The validation study showed the implemented EPT algorithm does not cope properly with 'condition blocking'. Since condition blocking is induced by the control framework it should not be a part of an EPT realization. However, current EPT algorithms are not able to cope with condition blocking. This should be resolved in future work. The validation study furthermore indicates that the other components satisfy the expectations and behave correctly with respect to the assumptions made.

The second step in the evaluation of the MPC framework compares the performance of this framework to the performance of a traditional approach (MRP-C). The main results of these experiments can be summarized as follows. For a single instantiation, the LDM will not converge to an optimal solution. This indicates either a flaw in the implementation of the LDM or that the problem is too complex for the optimization toolbox of Matlab. Due to a lack of time this issue has not been resolved within this research project. This instantiation is defined by two conditions. The output of the LDM, obtained for instantiations where these two conditions are not met, seem to provide correct results. A rolling horizon scheme has been used for the remainder of the experiments.

The results obtained from the first simulation case illustrate that the accuracy of the resource capacity constraint plays a vital role in the performance of the control(led) system. The presented MPC framework reduces both the variability and the average of the number of backorders by maintaining a higher (security) WIP level.

The difference in performance between the presented MPC framework and MRP-C is clearly illustrated by the third (sine) experiment. The results of this experiment show that the difference in performance increases with increasing utilization. This difference is caused by the fact that the MRP-C approach neglect the influence of variability by assuming a linear capacity relation.

Conversely, the MPC framework deploys approximations of the actual non-linear behavior. The difference between these two approximations increases with increasing utilization. Consequently, the production targets issued by the MPC framework will better correspond to the actual behavior of the production system at high utilization levels.

Since the first case did not contain a re-entrant product-flow, a second case is used to cover resource capacity partitioning. Analysis of the output of the LDM illustrates an unequal partitioning of WIP over the two competing steps at a workstation. Individually, the two steps meet the individual capacity requirements. However, additional WIP required to meet the total capacity is allocated with respect to the cost definition, rather than the expected equal distribution.

The implication of this unequal WIP distribution, is somewhat contradictory. Due to the unequal distribution of WIP, process time variability c_e^2 will have a larger influence on the (variation of the) output of the production system. On the other hand, the unequal WIP distribution decreases the effects caused by the variability in interarrival times c_a^2.

Since, for these simulation cases, EPT experiments show that $c_a^2 > c_e^2$, the effect of the unequal WIP distribution is positive with regard to the (variation of the) output of the production system. Note that, due to the linear capacity constraint of MRP-C, WIP is distributed equally over the production steps.

Simulation results of the second case illustrate the implication of the unequal WIP distribution. These results show a significant difference between the presented control framework and MRP-C. This difference is partially attributable to the unequal WIP distribution. Within the system controlled by the MPC framework, the effect of the interarrival time variability has been reduced significantly.

The results from the simulation experiments indicate that the presented MPC framework performs better than MRP-C by ensuring a more stable (lower mean and variation in the number of backorders) output of the production system. However, due to the convergence issue of the LDM an unambiguous performance evaluation cannot be obtained.

Chapter 6

Conclusions

So far, no satisfactory control framework for re-entrant production facilities has been presented. In this thesis, a two layer hierarchical MPC framework for re-entrant production facilities has been introduced. The performance of the MPC framework is analyzed with the use of a simulation model of a characteristic semiconductor production system. The simulation results are compared to that of a commonly used approach (MRP-C).

In this chapter, the most relevant findings of this research project are summarized.

Control framework

The literature survey illustrates that the majority of research focusses on single level approaches. Research on layered control frameworks, combining different approaches at their intended level, is rare. Furthermore, the survey illustrates that commonly applied high level control approaches assume fixed or linear capacity relations. Therefore, the achievable production capacity is commonly overestimated and inaccurate or infeasible production targets are issued.

In this thesis, a two layer hierarchical MPC framework is presented. The main objective of this control framework is to translate actual demand into controllable events for a re-entrant production system. This objective is achieved in two steps (layers).

At the high level control layer, production targets are derived based on the aggregated system state and on the predicted behavior of the production system (Model Predictive Control). Accurate system dynamics are obtained by implementing resource capacity constraints based on effective clearing functions (approximations from queuing theory). Essential parameters, required for the definition of the effective clearing functions, are determined by an EPT algorithm. The EPT algorithm effectively captures all sources of variability into a set of parameters.

An advantage of the presented high level control layer is that it circumvents the fundamental circularity present in many other production planning approaches.

Furthermore, by implementing a more accurate description of the non-linear resource dynamics within the Linear Discrete Model (LDM), the control system predicts flow times and resource capacity more accurately than the predictions obtained by the common approaches. In turn, the increased accuracy of the predictions results in improved production targets.

At the low level control layer, a composite dispatching rule is used to determine which lot (or batch) to process (dispatch) next. First, flow rate control determines the possible work to dispatch next, based on the issued production targets. Next, a 'Least Slack per remaining process step' (LS/n) sequencing policy is used to determine the optimal work to dispatch next. The LS/n policy is used because it has been proven to be stable and effective in a complex re-entrant production environment.

Performance

The presented MPC framework has been implemented in two simulation cases based on the Intel case. The first case is a three step flow line. It is used to evaluate the performance of the control framework without resource capacity partitioning. The second case resembles the original 'Intel Case' and is used to evaluate the performance in a re-entrant environment (thus requiring resource capacity partitioning).

Prior to the execution of the experiments, the different parts of the control framework were validated. The results of these validation experiments illustrate that the controlled system is subject to condition blocking, which has an undesired influence on the EPT measurements. Additional EPT quantification experiments have been conducted to circumvent condition blocking and to determine the parameters required for the definition of the effective clearing functions.

Analysis of the output of the LDM yielded a single cause of undesired control behavior. For a single instantiation, the LDM will not converge to an optimal solution. This indicates either a flaw in the implementation of the LDM or that the problem is too complex for the optimization toolbox of Matlab. Close analysis of the formal definition of both LDM's indicates that it is likely to be a flaw in the implementation rather than the formal definition. Due to the limited duration of the project, the convergence issue has not been resolved.

To circumvent the convergence issue, the simulation experiments are conducted in a rolling horizon scheme. To place the performance of the MPC framework into perspective, the simulation results are compared to that of similar experiments conducted with the MRP-C approach. The main results of these simulation experiments can be summarized as follows.

The simulation results indicate that, in general, the presented MPC framework outperforms MRP-C in the sense that the enforced demand trajectory is followed closely without creating a (permanent) backlog. Both the mean and the variability in the number of backorders are less for the presented control framework than for MRP-C.

The results of the sine experiment illustrate that the difference in performance increases with increasing utilization. This difference is caused by the fact that the MRP-C approach assumes a linear capacity relation. Conversely, the MPC framework deploys approximations of the actual non-linear behavior. The difference between these two approximations increases with increasing utilization. Consequently, the production targets issued by the MPC framework will better correspond to the actual behavior of the production system at high utilization levels.

However, the results of the second case are somewhat distorted by the unexpected WIP distribution that can be attributed to the resource capacity partitioning approach. Under the deployed approach, additional WIP required to meet the total capacity is allocated with respect to the cost definition, rather than the expected equal distribution. Since EPT quantification experiments show that for all workstations $c_a^2 > c_e^2$, the effect of the unequal WIP distribution (partitioning) is positive with regard to the (variation of the) output of the production system. Note that due to the linear capacity constraint of MRP-C, WIP is distributed equally over the production steps.

In general, the presented MPC framework results in a more predictable output (lower mean and variation in backorders) of the production system. The reduction in variation is procured by maintaining a (security) WIP level corresponding to the observed variability. However, a tradeoff will be made with regard to the mean total flowtime and the mean WIP level.

Chapter 7

Recommendations

During this research project, several questions have come up that could not be resolved due to the limited duration of the project. Based on these questions, recommendations for future research are formulated.

General assumptions

The control framework has been formulated based on a number of assumptions and simplifications. One of the principal assumptions is the use of a Linear Discrete Model for target setting. Although linear models are conceptually simple and often computationally tractable, the choice between linear models and non-linear models also depends on the number of variables, constraints and the performance of their solvers. Non-linear application of the discrete model would render the linearization algorithm obsolete and would enable the possibility to use the 'ideal' resource capacity partitioning approach. Consequently, the number of (resource capacity) constraints would be reduced significantly; respectively 1 resource capacity constraints instead of 10 linear approximations and n_k partitioned constraints per resource instead of $2^{n_k} - 1$. Therefore, the difference (in performance) between linear and non-linear application of the discrete model should be explored further.

Another principal assumption underlying the Linear Discrete Model is the real valued output. The results of the simulation experiments show a disrupting influence on the performance (partially) attributable to the real valued production targets issued by the LDM. It should be investigated how this issue can be solved. A solution might be to issue integer targets for lot releases only, as used by the MPC approach of Vargas-Villamil et al. (2003). During future research, the influence of (mixed-) integer solving on performance of the control framework should be investigated.

Effective Process Time algorithm

The results of the validation study illustrate that the controlled production system is subject to condition blocking. Since condition blocking has an undesired influence on the determined EPT parameters, an approach should be used to cope with condition blocking. However, current EPT algorithms are unable to handle condition blocking properly. Other controlled (complex) production systems are likely to be subject to condition blocking. Therefore, further research to develop an EPT algorithm that is able to properly handle condition blocking is advisable.

Resource capacity constraint

Within the LDM, accurate resource dynamics are obtained by implementing resource capacity constraints based on effective clearing functions (approximations from queuing theory). The idea to use effective clearing functions as a resource capacity constraint, is the major contributor to the performance of the MPC framework. The effective clearing function approach itself is applicable in other (MPC) control frameworks or even in current commercial software packages.

The simulation results of the second, re-entrant, case illustrate an unexpected behavior of the capacity partitioning approach. Although, the current constraints accurately describe the required wip for each individual step, additional WIP required for the total capacity is allocated with respect to the cost definition. The expected behavior was an equal partitioning over the individual process steps.

The capacity partitioning approach used here does not distinguish different products or steps consuming a resource at different rates. Furthermore, the current capacity partitioning approach requires $2^{n_k} - 1$ constraints per resource. For the second case, this approach resulted in, on average, 30 resource capacity constraints per workstation per time period. Clearly, deploying such a partitioning approach to a full-scale production environment would yield an explosion of the number of resource capacity constraints.

To effectively utilize the potential of the effective clearing functions, the influence of different partitioning approaches, as well as different WIP measures, should be explored. An alternative to reduce the number of resource constraints is to focuss attention only on (near-) bottleneck stations since these have the largest influence (Lu et al. 1994).

General performance

The results from the simulation experiments indicate that the presented MPC framework performs better than a traditional approach (MRP-C). However, due to the convergence issue of the LDM an unambiguous performance evaluation was not obtained. Therefore, it is recommended that the performance of the presented MPC framework is compared to several commonly encountered control approaches but also several newly

developed control approaches such as the MPC framework of Vargas-Villamil et al. (2003).

The advantages or extra capabilities of the presented MPC framework and those of the other control approaches should be evaluated. This evaluation would yield valuable information necessary in the search for a satisfactory control framework for re-entrant production facilities.

Bibliography

Adams, J., Balas, E. and Zawack, D. (1988), 'The shifting bottleneck procedure for job shop scheduling', *Management Science* **34**(3), 391–401.

Asmundsson, J., Rardin, R. L. and Uzsoy, R. (2003), Tractable nonlinear capacity models for aggregate production planning. working paper.

Bai, X., Srivatsan, N. and Gershwin, S. B. (1990), Hierarchical real-time scheduling of a semiconductor fabrication facility, *in* 'Ninth IEEE/CHMT International Electronic Manufacturing Technology Symposium. Competitive Manufacturing for the Next Decade. Proceedings 1990 IEMT Symposium', Washington DC, USA, pp. 312–317.

Blackstone, J. H., Philips, D. T. and Hogg, G. L. (1982), 'A state-of-the-art survey of dispatching rules for manufacturing job shop operations', *International Journal of Production Research* **20**(1), 27–45.

Campen, van, E. J. J. (2001), Design of a Multi-Process Multi-Product Waferfab, Phd thesis, Engineering Mechanics, Eindhoven University of Technology.

Chen, H., Harrison, J. M., Mandelbaum, A., van Ackere, A. and Wein, L. M. (1988), 'Empirical evaluation of a queueing network model for semiconductor wafer fabrication.', *Operations Research* **36**(2), 202–215.

Chern, C. C. and Liu, Y. L. (2003), 'Family-based scheduling rules of a sequence-dependent wafer fabrication system', *IEEE Transactions on Semiconductor Manufacturing* **16**(1), 15–25.

Coleman, T., Branch, M. A. and Grace, A. (1999), *Optimization Toolbox, For Use with MATLAB*, third edn, The MathWorks, Inc., Natick, MA. http://www.mathworks.com.

Conway, R. W., Maxwell, W. L. and Miller, L. W. (1967), *Theory of scheduling*, Addison-Wesley, Reading, MA, USA.

Dai, J. G. and Weiss, G. (1996), 'Stability and instability of fluid models for re-entrant lines', *Mathematics of Operations Research* **21**, 114–135.

Fargher, H. E., Kilgpre, M. A., Kleine, P. J. and Smith, R. A. (1994), 'A planner and scheduler for semiconductor manufacturing', *IEEE Transactions on Semiconductor Manufacturing* **7**(2), 117–126.

Fargher, H. E. and Smith, R. A. (1994), Planning in a flexible semiconductor manufacturing environment, *in* M. Zweben and M. Fox, eds, 'Intelligent Scheduling', Morgan Kaufman, San Francisco, pp. 545–581.

Fowler, J. W., Hogg, G. L. and Mason, S. J. (2002), 'Workload control in the semiconductor industry', *Production planning and control* **13**(7), 568–578.

Gershwin, S. B. (1989), 'Hierarchical flow control: a framework for scheduling and planning discrete events in manufacturing systems', *Proceedings of the IEEE* **77**(1), 195–208.

Gershwin, S. B., Akella, R. and Choong, Y. F. (1985), 'Short–term production scheduling of an automated manufacturing facility', *IBM Journal of Research and Development* **29**(4), 392–400.

Glassey, C. R. and Petrakian, R. G. (1989), *The use of bottleneck starvation avoidance with queue predictions in shop floor control*, University of California, Berkeley, CA, USA.

Glassey, C. R. and Resende, M. G. C. (1988), 'Closed-loop job release control for VLSI circuit manufacturing', *IEEE Transactions on Semiconductor Manufacturing* **1**(1), 36–46.

Graves, S. C. (1985), 'A tactical planning model for a job shop', *Operations Research* **34**, 552–533.

Habenicht, I. and Mönch, L. (2002), A finite-capacity beam-search-algorithm for production scheduling in semiconductor manufacturing, *in* E. Yücesan, C. H. Chen, J. L. Snowdon and J. M. Charnes, eds, 'Proceedings of the 2002 Winter Simulation Conference', San Diego, California, USA, pp. 1406–1413.

Hackman, S. T. and Leachman, R. C. (1989), 'A general framework for modeling production', *Management Science* **35**(4), 478–495.

Hadavi, K. and Voigt, K. (1994), A real time production scheduling system from conception to practice, *in* M. Zweben and M. Fox, eds, 'Intelligent Scheduling', Morgan Kaufman, San Francisco, pp. 581–604.

Hofkamp, A. T. and Rooda, J. E. (2002), χ *Reference Manual*, Systems Engineering Group, Eindhoven University of Technology. http://se.wtb.tue.nl.

Hopp, W. J. and Spearman, M. L. (2000), *Factory physics: foundations of manufacturing management*, second edn, McGraw-Hill, Boston, Massachusetts, USA.

Horiguchi, K., Raghavan, N., Uzsoy, R. and Venkateswaran, S. (2001), 'Finite-capacity production planning algorithms for a semiconductor wafer fabrication facility', *International Journal of Production Research* **39**(5), 825–842.

Hung, Y. F. and Leachman, R. C. (1996), 'A production planning methodology for semiconductor manufacturing based on iterative simulation and linear programming calculations', *IEEE Transactions on Semiconductor Manufacturing* **9**(2), 257–269.

Jacobs, J. H., Etman, L. F. P., van Campen, E. J. J. and Rooda, J. E. (2003), 'Characterization of operational time variability using effective process time', *IEEE Transactions on Semiconductor Manufacturing* **16**(3), 511–520.

Johnson, L. A. and Montgomery, D. C. (1974), *Operations Research in Production Planning, Scheduling, and Inventory Control*, Wiley, New York, New York, USA.

Karmarkar, U. S. (1989), 'Capacity loading and release planning with work-in-progress (wip) and lead-times', *Journal of Manufacturing and Operations Management* **2**, 105–123.

Kempf, K. G. (2003), 'Intel five-machine six step mini-fab description'. http://www.eas.asu.edu/~aar/research/intel/papers/fabspec.html.

Kimemia, J. and Gershwin, S. B. (1983), 'An algorithm for the computer control of a flexible manufacturing system', *IIE Transactions* **15**(4), 353–362.

Kleijnen, J. P. C. (1995), 'Verification and validation of simulation models', *European Journal of Operational Research* **82**(1), 145–162.

Kock, A. A. A. (2003), Performance evaluation and simulation meta modeling of single server flow lines subject to blocking: an effective process time approach, Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands. SE420367.

Kumar, P. R. (1993), 'Re-entrant lines', *Queueing Systems: Theory and Applications, Special Issue on Queueing Networks* **13**(1-3), 87–110.

Leachman, R. C. (2001), Semiconductor production planning, *in* P. M. Pardalos and M. G. C. Resende, eds, 'Handbook of Applied Optimization', New York, New York, USA, pp. 746–762.

Leachman, R. C., Solorzano, M. and Glassey, C. R. (1988), *A queue management policy for the release of factory work orders*, University of California, Berkeley, CA, USA.

Liao, D. Y., Chang, S. C., Pei, K. W. and Chang, C. M. (1996), 'Daily scheduling for R&D semiconductor fabrication', *IEEE Transactions on Semiconductor Manufacturing* **9**(4), 550–561.

Little, J. D. C. (1961), 'A proof of the formula $l = \lambda \cdot w$', *Operations Research* **9**, 383–387.

Lou, S. X. C. and Kager, P. W. (1989), 'A robust control policy for VLSI wafer fabrication', *IEEE Transactions on Semiconductor Manufacturing* **2**(4), 159–164.

Lu, S. C. H., Ramaswamy, D. and Kumar, P. R. (1994), 'Efficient scheduling policies to reduce mean and variance of cycle-time in semiconductor manufacturing plants', *IEEE Transactions on Semiconductor Manufacturing* **7**(3), 374–388.

Lu, S. H. and Kumar, P. R. (1991), 'Distributed scheduling based on due dates and buffer priorities', *IEEE Transactions on Automatic Control* **36**(12), 1406–1416.

Lutz, M. and Ascher, D. (1999), *Learning Python*, first edn, O'Reilly & Associates, Inc, Sebastopol.

Mason, S. J. and Fowler, J. W. (2000), Maximizing delivery performance in semiconductor wafer fabrication facilities, *in* J. A. Joines, R. R. Barton, K. Kang and P. A. Fishwick, eds, 'Proceedings of the 2000 Winter Simulation Conference', Orlando, Florida, USA, pp. 1458–1463.

Pinedo, M. (2001), *Scheduling: Theory, Algorithms, and Systems*, second edn, Prentice Hall, Englewood Cliffs, NJ.

Pollaczek, F. (1930), Uber eine aufgabe dev wahrscheinlichkeitstheorie, *in* 'I-II Math. Zeitschrift', Vol. 32, pp. 64–100, 729–750. (in German).

Sethi, S. P. and Zhang, Q. (1994), *Hierarchical Decision Making in Stochastic Manufacturing Systems*, Birkhauser Boston, Cambridge, MA, USA.

Shen, Y. and Leachman, R. C. (2003), 'Stochastic wafer fabrication scheduling', *IEEE Transactions on Semiconductor Manufacturing* **16**(1), 2–14.

Sohl, D. L. and Kumar, P. R. (1995), Fluctuation smoothing scheduling policies for multiple process flow fabrication plants, *in* 'Seventeenth IEEE/CPMT International Electronics Manufacturing Technology Symposium', Austin, Texas, USA, pp. 190–198.

Spearman, M. L., Woodruff, D. L. and Hopp, W. J. (1990), 'Conwip: a pull alternative to kanban', *International Journal of Production Research* **28**(5), 879–894.

Srinivasan, A., Carey, M. and Morton, T. A. (1988), Resource pricing and aggregate scheduling in manufacturing systems. GSIA, Carnegy-Mellon University.

Sterian, A. (1999), 'Pymat - an interface between python and matlab'. http://claymore.engineer.gvsu.edu/~steriana/Python/pymat.html.

Tardif, V. and Spearman, M. L. (1997), 'Diagnostic scheduling in finite-capacity production environments', *Computers and Industrial Engineering* **32**(4), 867–878.

Thomas, Jr., G. B. and Finney, R. L. (1996), *Calculus and Analytic Geometry*, ninth edn, Addison-Wesley Publishing Company.

Toba, H. (2000), 'Segment-based approach for real-time reactive rescheduling for automatic manufacturing control', *IEEE Transactions on Semiconductor Manufacturing* **13**(3), 264–272.

Tsakalis, K. S., Flores-Godoy, J. J. and Rodriguez, A. A. (1997), Hierarchical modeling and control of re-entrant semiconductor fabrication lines: A mini-fab benchmark, *in* '6th IEEE Intl. Conf. on Emerging Technologies and Factory Automation', Los Angeles, California, USA, pp. 508–513.

Uzsoy, R., Lee, C. Y. and Martin-Vega, L. A. (1992), 'A review of production planning and scheduling models in the semiconductor industry; part I: System characteristics, performance evaluation and production planning', *IIE Transactions* **24**(4), 47–60.

Uzsoy, R., Lee, C. Y. and Martin-Vega, L. A. (1994), 'A review of production planning and scheduling models in the semiconductor industry; part II: Shop-floor control', *IIE Transactions* **26**(5), 44–55.

Uzsoy, R., Martin-Vega, L. A., Lee, C. Y. and Leonard, P. A. (1991), 'Production scheduling algorithms for a semiconductor test facility', *IEEE Transactions on Semiconductor Manufacturing* **4**(4), 270–280.

Vargas-Villamil, F. D. and Rivera, D. E. (2001), 'A model predictive control approach for real-time optimization of re-entrant manufacturing lines', *Computers in Industry* **45**(1), 45–57.

Vargas-Villamil, F. D., Rivera, D. E. and Kempf, K. G. (2003), 'A hierarchical approach to production control of re-entrant semiconductor manufacturing lines', *IEEE Transactions on control systems technology* **11**(4), 578–587.

Vollmann, T. E., Berry, W. L. and Whybark, D. C. (1997), *Manufacturing Planning and Control Systems*, fourth edn, McGraw-Hill/Irwin, Homewood, Illinois, USA.

Waikar, A. M., Sarker, B. R. and Lal, A. M. (1995), 'A comparative study of some priority dispatching rules under different shop loads', *Production planning and control* **6**(4), 301–310.

Weber, S. (2003), Design of real-time supervisory control systems, Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands. SE420330.

Wein, L. M. (1988), 'Scheduling semiconductor wafer fabrication', *IEEE Transactions on Semiconductor Manufacturing* **1**(3), 115–130.

Weiss, G. (1999), 'Scheduling and control of manufacturing systems — a fluid approach', *Proceedings of the 37th Allerton Conference* pp. 577–586.

Appendix A

Effective clearing functions

The complete derivation of the effective clearing function, from queuing theoretic approximation to resource capacity constraint, is described in this Appendix. The effective clearing function will be used as a resource capacity constraint in the MPC framework, introduced in Chapter 3.

A.1 Queuing theoretic approximation

Consider a general workstation with m parallel machines, batch-size b and general inter-arrival and process times ($G/G^b/m$), illustrated by Figure A.1. Arriving lots are stored in a buffer, until a batch is complete and a machine runs idle. The physical buffer, in front of the workstation, can be divided into two sequential (virtual) buffers; a batching buffer B_b and a queueing buffer B_q. Processing of the individual parts of a batch, is carried out in parallel, i.e. $t_{e,b} = t_{e,l}$.

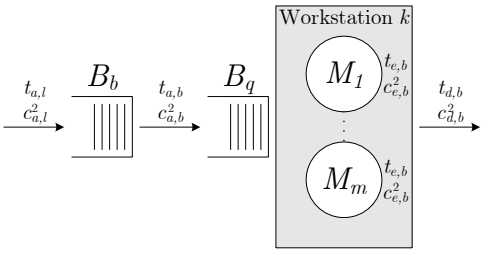


Figure A.1: $G/G^b/m$ workstation

The mean total flow time of this workstation $\varphi(G/G^b/m)$ consists of the mean wait-in-batch-time φ_b, the queuing time φ_q and the effective process time t_e, (A.1).

$$\varphi(G/G^b/m) = \varphi_b(G/G^b/1) + \varphi_q(G/G^b/m) + t_e. \tag{A.1}$$

The mean wait-to-batch-time φ_b is not equal for all lots in a batch. The first lot in the batch has to wait for $(b-1)$ lots to arrive, while the last lot does not have to wait at all before the batch is complete. The mean wait-to-batch-time φ_b for the complete batch is equal to the average of these two extremes (A.2).

$$\varphi_b(G/G^b/1) = \frac{b-1}{2} \cdot \frac{t_e}{bu} = \frac{b-1}{2\delta} \tag{A.2}$$

The mean queuing time φ_q of a $G/G^b/m$ workstation (A.3) is approximated (Hopp and Spearman 2000) by multiplying a variability term with the mean queuing time of a workstation with exponential distributed times $\varphi_q(M/M/m)$. Here, $\varphi_q(G/G^b/m)$ is defined by (A.4). Note that, (A.4) is adapted for batch-processing by assuming $c^2_{a,b} = c^2_{a,l}/b$.

$$\varphi_q(G/G^b/m) = \frac{c^2_{a,l}/b + c^2_e}{2} \cdot \varphi_q(M/M/m) \tag{A.3}$$

$$\varphi_q(M/M/m) = \frac{u^{\sqrt{2(m+1)}-1}}{m(1-u)} \cdot t_e \tag{A.4}$$

Combining (A.1) up to (A.4) results in an approximation of the mean total flow time of a $G/G^b/m$ workstation (A.5).

$$\varphi(G/G^b/m) = \frac{b-1}{2\delta} + c^2 \cdot \frac{u^{\gamma-1}}{m(1-u)} \cdot t_e + t_e \tag{A.5}$$

$$\text{where } c^2 = (c^2_a/b + c^2_e)/2 \text{ and } \gamma = \sqrt{2(m+1)}$$

With the use of Little's Law, $w = \delta \cdot \varphi$, (A.5) can be transformed into (A.6); describing the mean work-in-progress $w(G/G^b/m)$ as a function of the utilization u. Little's law is used under the assumption that the system is in a local steady-state.

$$w(G/G^b/m) = \frac{b-1}{2} + c^2 \cdot \frac{u^{\gamma-1}}{m(1-u)} \cdot t_e\delta + t_e\delta \tag{A.6}$$

The machine utilization u, describing the probability that the machine is busy, equals (A.7).

$$u = \frac{r_a}{r_e} = \frac{t_e\delta}{mb} \tag{A.7}$$

Applying (A.7) to (A.6) results in (A.8); an approximation of $w(G/G^b/m)$ as function of the δ. All other parameters of (A.8), e.g. t_e, are assumed to be constant.

$$w(G/G^b/m) = \frac{b-1}{2} + bc^2 \cdot \left(\frac{t_e}{mb}\right)^{\gamma} \cdot \frac{\delta^{\gamma}}{1 - \frac{t_e}{mb}\delta} + t_e\delta \tag{A.8}$$

In accordance with the definition of the effective clearing function, $\delta = f(w)$, (A.8) has to be solved for δ (A.9). However, due to the highly non-linear behavior, (A.8) can not analytically be solved for δ. Therefore, (A.8) is solved numerically with the use of e.g. the bisection method.

$$\delta(G/G^b/m) = f(w, c^2, t_e, m, b) \tag{A.9}$$

A.2 Linear approximation of the effective clearing function

To incorporate the effective clearing functions into a Linear Discrete Model, the functions have to be captured by a set of linear approximations, i.e. $\delta = \boldsymbol{\alpha} \cdot w + \boldsymbol{\beta}$ recall (3.5). Here, $\boldsymbol{\alpha}$ denotes the set of gradient parameters and $\boldsymbol{\beta}$ denotes the set of scaling parameters. The effective clearing function is consequently defined by (A.10).

$$f(w_t) = \min_{\forall c} \left\{ \alpha_t^c \cdot w_t + \beta_t^c \right\} \tag{A.10}$$

Although it is not possible to solve (A.8) analytically, it can be used to determine the derivative of (A.9). Consequently, (A.11) defines the derivative of (A.9) as the inverse of the derivative of (A.8). The scaling parameter β is determined by solving the linearized function, $\delta = \alpha w + \beta$, at the point of linearization; $(w_\star, \delta_\star)$ (A.12).

$$\alpha = \frac{\partial \delta}{\partial w} = \frac{\partial w}{\partial \delta}^{-1} = \left\{ c^2 b \cdot \frac{t_e}{mb}^{\gamma} \cdot \left(\frac{\gamma \delta^{\gamma-1}}{1 - \frac{t_e}{mb}\delta} + \frac{\frac{t_e}{mb}\delta^{\gamma}}{(1 - \frac{t_e}{mb}\delta)^2} \right) + t_e \right\}^{-1} \tag{A.11}$$

$$\beta = \delta_\star - \alpha w_\star \tag{A.12}$$

Due to the dynamical typesetting, the effective clearing functions should be redetermined each period. Consequently, a linearization algorithm is implemented in the high level control layer to determine the set of linear approximations $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$.

Linearization algorithm The objective of the linearization algorithm is to derive a set of linear approximations, $\{\boldsymbol{\alpha},\boldsymbol{\beta}\}$ required by (3.5). The linearization algorithm performs the following steps to determine $\{\boldsymbol{\alpha},\boldsymbol{\beta}\}$:

1. The first linear approximation will be placed at the Origin, $(0, 0)$, resulting in (α_0, β_0).
2. The next approximation will be placed at the point where the difference, in δ, between the former linearization and the actual function is equal to a redescribed percentage, p_δ. Iteration continues until the approximation of the gradient is less than the tolerance, i.e $\alpha \leq \alpha_{tol}$.
3. The last linear approximation will be the one for $\lim_{w\to\infty} \delta(G/G/m)$, i.e. $(0, \beta_{\delta\to\delta_m}) = (0, \delta_m)$ ◀

The linearization algorithm is illustrated by Example A.1

Example A.1 Consider a workstation with two (identical) parallel machines, a squared coefficient of variation c^2 of 2.5 and a mean process time of 5. For this $G/G/2$-workstation, the non-linear 'effective clearing function' (3.4) can be derived numerically.

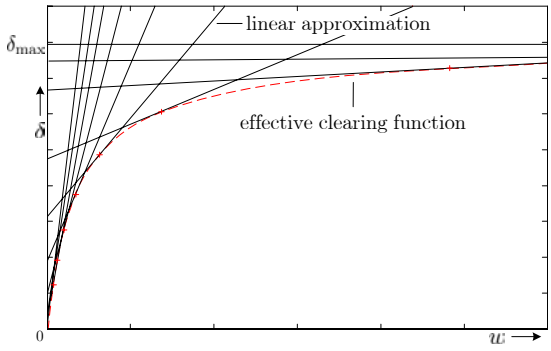


Figure A.2: Set of linear approximations capturing the effective clearing function.

Linearization of the effective clearing function yields a set of 10 linear approximations $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$. Figure A.2 illustrates the linear approximation of the effective clearing function for a $G/G/2$ workstation. ◀

A.3 Newton-Raphson method

By using the derivative of (A.9), it is possible to use the Newton-Raphson (NR) method to numerically solve (A.8) for δ. Advantage of the Newton-Raphson above the bisection method, is that it is able to solve with a second order convergence speed, compared to the first order of the bisection-method.

Newton-Raphson method The goal of Newton-Raphson's method for estimating a solution of an equation $f(x) = 0$ is to produce a sequence of approximations that approach the solution, by using the following strategy:

1. Guess a first approximation to a root of the equation $f(x) = 0$.
2. Use the first approximation to obtain a second, the second to get a third and so on, using formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad (f'(x_n) \neq 0)$$

where $f'(x_n)$ is the derivative of f at x_n.

Convergence is assured if, for any value of x, the following equation holds

$$\left| \frac{f(x)f''(x)}{[f'(x)]^2} \right| < 1$$

for all x in an interval about a root r. (Thomas, Jr. and Finney 1996) ◀

In case of (A.8), convergence of the NR method, for any starting value δ_0, is guaranteed due to the convex nature of the function.

Appendix B

Simulation framework

In this appendix, the simulation framework, with which the presented MPC framework is evaluated, is discussed. In the first section, a detailed description of the different modules of the simulation framework is presented. The code of the case-independent Python modules is presented in Section B.1. The case dependent modules are discussed in Appendix C and D for respectively the first and second case.

The complete set of files that were used for the implementation and evaluation of the MPC framework are included on the CD-ROM, accompanying this thesis.

B.1 Framework structure

As mentioned in Section 4.1, both the Discrete Event Simulation Model (DESM) as well as the distributed low level control layer are implemented in χ-0.8. Whereas, the high level control layer of the MPC framework is implemented in Matlab. The general form of the simulation framework is introduced in Section 4.1. A more detailed representation of the simulation framework is presented by Figure B.1.

The heart of the simulation framework is the (compiled) 'χ-model'; the actual Discrete Event Simulation Model. At the end of each shift, various functions in the χ-model call the Matlab functions to solve the optimization problem (LDM) and retrieve the 'optimal' production targets.

The interaction between the χ-model and the Matlab functions is achieved by deploying the Pymat-interface (Sterian 1999). Within Python, the module 'io.ext' connects the data-input to the χ-model, for which the module 'data.py' and 'setup.py' are used. The module 'data.py' defines several functions that are needed for the data-handling of the χ-models. It furthermore imports the dictionary defined in 'setup.py'. This dictionary contains the relevant simulation parameters (such as, the step process times, initial condition of the production system etc.).

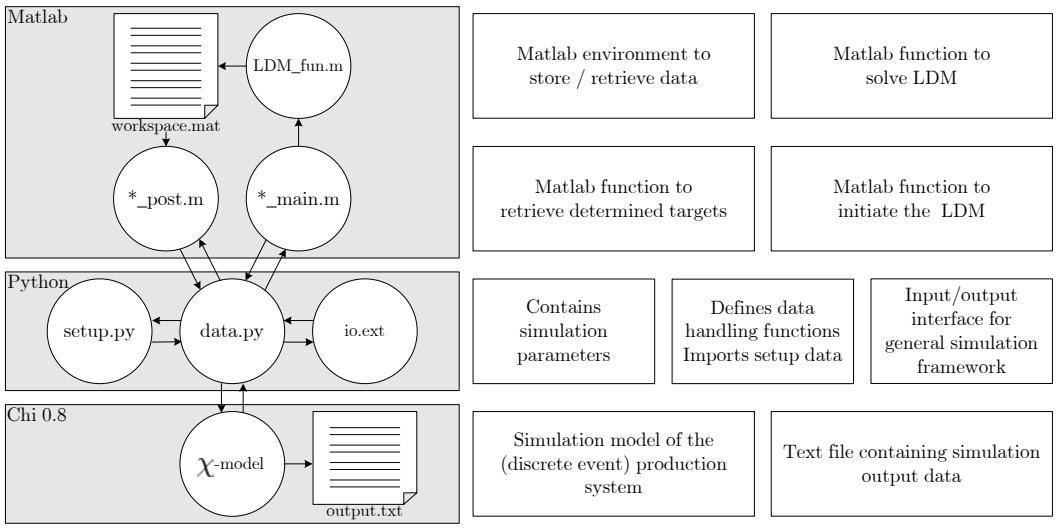


Figure B.1: Schematic representation of a simulation framework

The 'data.py' module is moreover used for the initialization of the essential parameters in the Matlab environment.

Within the Matlab environment, the pre-optimization parameter transformations are handled by '∗_main.m'. The various matrices (recall (3.10)) defining the (case dependent) LDM, are initiated by 'LDM_fun.m'. The Matlab functions defining the (matrices of the) LDM are discussed in Appendix C and D, for respectively the first and the second case.

After initiation of the matrices, 'LDM_fun.m' calls the solver of the 'optimization toolbox' to solve the optimization problem. Subsequently, 'LDM_fun.m' retrieves the output of the LDM and stores it in the 'workspace.mat'-file. Triggered by the simulation model, '∗_post.m' retrieves the output and transforms it into χ-compliant production targets.

In the next section, the case independent Python code is presented. The (case dependent) code of the two simulation cases is presented in respectively Appendix C and Appendix D.

B.2 Python code

In this section, the code of the python modules, discussed in the previous section, is presented. The interaction between the different modules is illustrated by Figure B.1.

io.ext

```
language "python"
```

```
file "data"

ext GetNat(s: string)   -> nat                                          = "Get"
ext GetReal(s: string)  -> real                                         = "Get"
ext GetString(s: string)-> string                                       = "Get"

ext GetTe(s: string)    -> (real^3)^6                                   = "Get"
ext GetTeEst(s: string) -> (real^3)^3                                   = "Get"
ext GetTo(s: string)    -> (real^2)^6                                   = "Get"
ext GetTs(s: string)    ->  real^4                                      = "Get"
ext GetBs(s: string)    ->  nat^3                                       = "Get"
ext Gettg(s: string)    -> (real#(real)^6#(real)^6#(real)^6#real#real) = "Get"
ext GetWini(s: string)  -> (nat#nat#nat#real#real)*^3                   = "Get"

ext GetAp(s: string)    -> string#nat#real                              = "Get"
ext GetDp(s: string)    -> string#nat#nat#nat#nat                       = "Get"
ext GetFc(s: string)    -> real#nat#nat#nat#nat#nat                     = "Get"

ext LPsolve( ns,np,p: nat
           , W: nat^7
           , ept: (real*^2)^3
           , est: (real^3)^3
           , Ap: (string#nat#real)
           , Cp: string
           , Dp: (string#nat#nat#nat#nat)
           , Ep: string
           , Tp: string
           , Er: real
           , BO: real
           , Fc: (real#nat#nat#nat#nat#nat)
           , dbug: nat
           )-> bool

ext LPpost( t: real
          , r,p: nat
          , W: nat^7
          , Er: real
          , BO: real
          )-> (real#(real^6)#(real^6)#(real^6)#real#real)
```

data.py

```
# data.py
from Numeric import *
import setup, pymat, sys, os, string, math

# 1. Initialisation
dict = setup.par

# 2. Get functions
def Get(s):
    return dict[s]

# 3. Solving LP model with the use of Matlab
def LPsolve(ns, np, p, W, ept, est, Ap, Cp, Dp, Ep, Tp, Er, BO, Fc, dbug):

    # 3.a Local parameter declaration
    i_ns     = array([ns])
    i_np     = array([np])
    i_p      = array([p])
```

```
i_dbug   = array([dbug])
i_Atype  =        Ap[0]
i_Ahor   = array([Ap[1]])
i_Aalpha = array([Ap[2]])
i_Ctype  = Cp
i_Dtype  =        Dp[0]
i_Dini   = array([Dp[1]])
i_Dper   = array([Dp[2]])
i_Dmin   = array([Dp[3]])
i_Dmax   = array([Dp[4]])
i_Etype  = Ep
i_Ttype  = Tp
i_Erem   = array([Er])
i_BOini  = array([BO])
i_F      = array([Fc])
i_W      = array([W])
i_EaWS1  = transpose(array([(ept[0])[0]]))
i_EeWS1  = transpose(array([(ept[0])[1]]))
i_EaWS2  = transpose(array([(ept[1])[0]]))
i_EeWS2  = transpose(array([(ept[1])[1]]))
i_EaWS3  = transpose(array([(ept[2])[0]]))
i_EeWS3  = transpose(array([(ept[2])[1]]))
i_estWS1 = array([est[0]])
i_estWS2 = array([est[1]])
i_estWS3 = array([est[2]])

# 3.b Matlab initialization
H = pymat.open()
pymat.eval(H,"clear all")
pymat.eval(H,"pth = pwd")
pymat.eval(H,"cd(strcat(pth,'/Matlab/'))")

if dbug == 1:
             pymat.eval(H,"diary")

# 3.c Matlab parameter declaration
pymat.put(H,"ns"    , i_ns)
pymat.put(H,"np"    , i_np)
pymat.put(H,"p"     ,  i_p)
pymat.put(H,"dbug"  , i_dbug)
pymat.put(H,"Atype" , i_Atype)
pymat.put(H,"Ahor"  , i_Ahor)
pymat.put(H,"Aalpha", i_Aalpha)
pymat.put(H,"Ctype" , i_Ctype)
pymat.put(H,"Dtype" , i_Dtype)
pymat.put(H,"Dini"  , i_Dini)
pymat.put(H,"Dper"  , i_Dper)
pymat.put(H,"Dmin"  , i_Dmin)
pymat.put(H,"Dmax"  , i_Dmax)
pymat.put(H,"Etype" , i_Etype)
pymat.put(H,"Ttype" , i_Ttype)
pymat.put(H,"Erem"  , i_Erem)
pymat.put(H,"BOini" , i_BOini)
pymat.put(H,"F"     , i_F)
pymat.put(H,"W"     , transpose(i_W))
pymat.put(H,"EaWS1" , i_EaWS1)
pymat.put(H,"EeWS1" , i_EeWS1)
pymat.put(H,"EaWS2" , i_EaWS2)
pymat.put(H,"EeWS2" , i_EeWS2)
pymat.put(H,"EaWS3" , i_EaWS3)
pymat.put(H,"EeWS3" , i_EeWS3)
pymat.put(H,"estWS1", i_estWS1)
```

```
    pymat.put(H,"estWS2", i_estWS2)
    pymat.put(H,"estWS3", i_estWS3)

    # 3.d Matlab function call
    pymat.eval(H,"case_I_main")         # Solve the LDM
    pymat.eval(H,"save output.mat")     # Save output of LDM

    # 3.e Confirm completion to Chi
    return 1

# 4. Retrieve output of the LDM
def LPpost(t, r, p, W, Er, BO):

    # 4.a Local parameter declaration
    i_r     = array([r])
    i_W     = array([W])
    i_Erem   = array([Er])
    i_BOini  = array([BO])

    # 4.b Matlab initialization
    H = pymat.open()
    pymat.eval(H,"clear all")
    pymat.eval(H,"pth = pwd")
    pymat.eval(H,"cd(strcat(pth,'/Matlab/'))")

    # 4.c Matlab parameter declaration
    pymat.eval(H,"load output.mat")     # load LDM solution
    pymat.put(H,"r", i_r)

    # 4.d Matlab function call
    pymat.eval(H,"inline_post")         # Post-processing

    # 4.e Matlab parameter retrieval
    o_G  = pymat.get(H,"out_G")[0]
    o_X  = pymat.get(H,"out_X")
    o_W  = pymat.get(H,"out_W")
    o_C  = pymat.get(H,"out_C")
    o_E  = pymat.get(H,"out_E")[0]
    o_BO = pymat.get(H,"out_BO")[0]

    # 4.f Python write parameters / targets to output file
    if p == 1:
              output = open('./output/parameters.txt','w')
    else:
              output = open('./output/parameters.txt','a')
    output.write(str(t) + '\t' + str(p) + '\t')

    i = 0
    while i < len(i_W[0]):
          output.write(str((i_W[0])[i]) + '\t')
          i = i + 1
    output.write(str(i_Erem[0]) + '\t' + str(i_BOini[0]) + '\t' + str(o_G) + '\t')

    i = 0
    while i < len(o_X):
          output.write(str(o_X[i]) + '\t')
          i = i + 1
    i = 0
    while i < len(o_W):
          output.write(str(o_W[i]) + '\t')
          i = i + 1
    i = 0
```

```
    while i < len(o_C):
          output.write(str(o_C[i]) + '\t')
          i = i + 1
    output.write(str(o_E) + '\t' + str(o_BO) + '\n')
    output.close()

    # 4.g Close matlab
    pymat.close(H)

    # 4.h Return parameters to Chi
    return (o_G, tuple(o_X), tuple(o_W), tuple(o_C), o_E, o_BO)
```

setup.py

```
par = { 'n_ini':   4
      , 'dbug' :   0
      , 'ss'   : 720.0
      , 'ta'   :   5.0
      , 'tm'   :   0.8
      , 'ts'   : ( 0.0, 5.0, 10.0, 12.0)
      , 'te'   : ( (225.0, 0.0, 0.0)                  # Step 0 (te, ce2, ca2)
                 , ( 30.0, 0.0, 0.0)                  # Step 1 (te, ce2, ca2)
                 , ( 55.0, 0.0, 0.0)                  # Step 2 (te, ce2, ca2)
                 , ( 50.0, 0.0, 0.0)                  # Step 3 (te, ce2, ca2)
                 , (255.0, 0.0, 0.0)                  # Step 4 (te, ce2, ca2)
                 , ( 10.0, 0.0, 0.0) )                # Step 5 (te, ce2, ca2)
      ,'te_est': ( (225.0, 0.1, 0.1)                  # WS0 (te, ce2, ca2)
                 , ( 30.0, 0.1, 0.1)                  # WS1 (te, ce2, ca2)
                 , ( 55.0, 0.1, 0.1) )                # WS2 (te, ce2, ca2)
      , 'to'   : ( ( 20.0, 40.0)                      # Step 1
                 , ( 15.0, 15.0)                      # Step 2
                 , ( 10.0, 10.0)                      # Step 3
                 , ( 15.0, 15.0)                      # Step 4
                 , ( 20.0, 40.0)                      # Step 5
                 , ( 10.0, 10.0) )                    # Step 6
      , 'bs'   : ( 3, 1, 1 )
      , 'w_ini': ( [ (4,0,0, 3.0,3.0)
                   , (3,0,0, 2.0,2.0) ]               # Wip in WS0
                 , [(2,0,1, 1.0,1.0)]                 # Wip in WS1
                 , [(1,0,2, 0.0,0.0)]                 # Wip in WS2
                 )
      , 'tg_ini':( 12.0                               # Release target
                 , (4.0, 12.0, 12.0, 12.0, 4.0, 12.0) # X target
                 , (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)     # W target
                 , (310.0, 85.0, 55.0, 0.0, 0.0, 0.0) # C target
                 , 12.0                               # Exit target
                 , 0.0 )                              # initial BO
      , 'Fc'   : ( 0.5, 20, 0, 0, 0, 10000)           # Cost function (scw, cw, cr, cp, cd, cb)
      , 'Ap'   : ( 'average', 10, 0.5)                # Atype, Ahor, Aalpha
      , 'Cp'   : 'CF'                                 # Ctype
      , 'Ep'   : 'off'                                # Etype: online EPT measuring {on / off}
      , 'Tp'   : 'normal'                             # Ttype {normal,floor, ceil, round}
      , 'Dp'   : ( 'stepup', 10, 10, 0, 6)            # Dtype, Dini, Dper, Dmin, Dmax
      , 'T'    : 30                                   # Time horizon length
```

Appendix C

Case I Description

In this appendix, the files used during the implementation of the first case are described. A detailed description of the first case is presented in Section 4.3. This appendix, starts with a description of the case dependent Matlab functions.The χ-model of the first case is discussed in Section C.2.

C.1 Matlab functions

In this section, the Matlab functions, defining the Linear Discrete Model (LDM), are discussed. The general form of the Linear Discrete Model is formulated by (3.10). However, within the Matlab environment, a different denomination is used. Here, the matrices, defining the LDM, are denominated according to Coleman et al. (1999). Consequently, the altered general form of the LDM, with the Matlab-denomination is defined by (C.1). Note that the form corresponds to that of (3.10).

$$\min_{\mathbf{x}} \quad f^T \cdot \mathbf{x} \tag{C.1a}$$

$$\text{s.t.} \quad \mathbf{Aeq} \cdot \mathbf{x} = \mathbf{beq} \tag{C.1b}$$

$$\mathbf{Aineq} \cdot \mathbf{x} \leq \mathbf{bineq} \tag{C.1c}$$

$$\mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \tag{C.1d}$$

The heart of the Matlab-environment is 'LDM_fun.m'. This function initiates the different matrices by calling the corresponding functions. The majority of the function names correspond to the matrices the various functions populate. In addition, '$*$_capcon.m' encompasses the linearization algorithm (Appendix A and defines the sets of linear approximations capturing the effective clearing functions (3.5).

After initiation of the matrices and parameters, 'LDM_fun.m' calls the solver of the 'optimization toolbox' to solve the optimization problem. Subsequently, the output is stored into the 'workspace.mat'-file, where it awaits further processing. The interaction of the files, defining the LDM, is illustrated by Figure C.1.

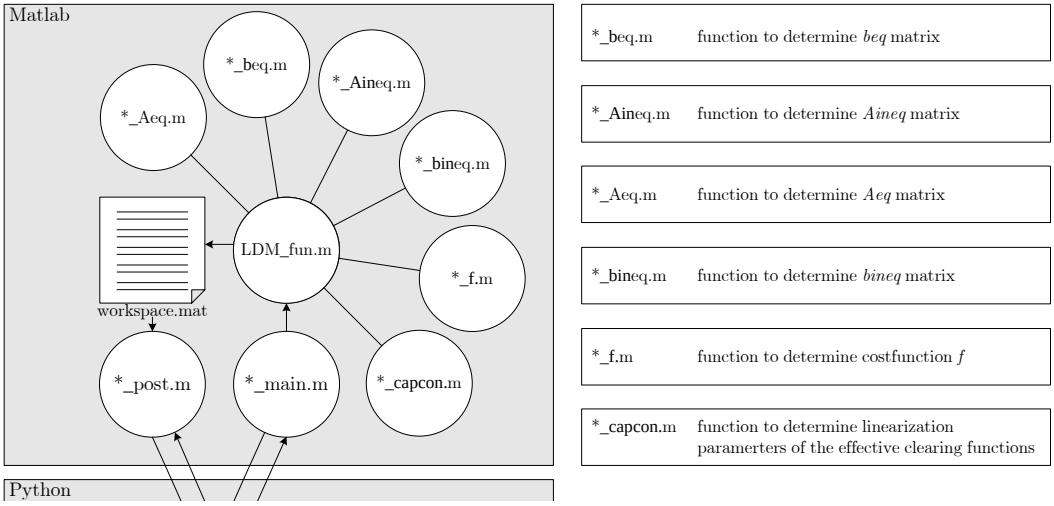


Figure C.1: Schematic representation of the Matlab functions initiating the LDM

Matlab code

In this section, the code of the case dependent Matlab functions, defining the LDM, is presented. The interaction between the different files is illustrated by Figure C.1.

case_I_standalone.m

```
ns     =   630;              % planning horizon
np     =     1;              % number of products
p      =     1;              % shift id
r      =     1;

dbug   =     0;              % dbug value

Atype  =    'average';       % Type of average calculation function {average, n-average, ewma}
Ahor   =    10;              % Horizon of realizations the average is based on
Aalpha =     0.5;            % Exponential Weighted Moving Average parameter

Ctype  =    'CF';            % Resource capacity constraint type {MRP, MRP-C, CF}

Dtype  =    'sinus';         % Type of demand function {stepup, stepdown, rampup, rampdown, sinus}
Dini   =    48;              % minimum value of demand function
Dper   =    96;              % maximum value of demand function
Dmin   =     0;              % Initialization period
Dmax   =    16;              % (Transition) Period length

Etype  =    'off';           % measure online EPT {on, off}

Ttype  =    'normal';        % Resource capacity constraint type {normal, floor, ceil, round}

Erem   =     0;              % Remains of the Exit target
BOini  =     0;              % initial value of back-orders

% Cost definition
```

```
F(1,1) =     0.5;           % Increase in inventory holding cost
F(1,2) =    20;             % Inventory holding cost
F(1,3) =     0;             % Release cost
F(1,4) =     0;             % Production cost
F(1,5) =     0;             % Demand cost
F(1,6) = 10000;             % Backorder cost

W = [1; 0; 0; 0; 0; 0; 0];   % Initial WIP value

EaWS1 = [];
EeWS1 = [];
EaWS2 = [];
EeWS2 = [];
EaWS3 = [];
EeWS3 = [];

estWS1 = [ 285  , 0.93, 11.0];
estWS2 = [  60  , 0.38, 4.54];
estWS3 = [  52.5, 0.57, 2.17];

Wmax   = [ 18 12 12 ];
```

case_I_main.m

```
load activeconstraints.mat                    % loading active constraints matrix

ss     = 12;                                  % shift size
sp     =  5;
sd     =  3;

x_dev  =  1.1;                                % Maximum allowed deviation of the former tangent
nm     = [2; 2; 1; 2; 2; 1];                  % number of resources per workstation
bs     = [3; 1; 1; 1; 3; 1];                  % batchsize per workstation
col    = [1];

W  = filter_W(W, Wmax);
D  = demand_gen(p, ns, Dtype, Dini, Dper, Dmin, Dmax);             % demand trajectory

pWS(1,1:3) = WS_par(EaWS1, EeWS1, estWS1, Atype, Ahor, Aalpha, Etype);
pWS(2,1:3) = WS_par(EaWS2, EeWS2, estWS2, Atype, Ahor, Aalpha, Etype);
pWS(3,1:3) = WS_par(EaWS3, EeWS3, estWS3, Atype, Ahor, Aalpha, Etype);
te = [ pWS(1,1); pWS(2,1); pWS(3,1); pWS(2,1); pWS(1,1); pWS(3,1)];

cfp = (te./60)./(nm.*ss);

% Effective linear clearing functions
for i = 1 : 3                                 % Workstations
    [Alpha,Beta] = ECF_fun_capcon( pWS(i,3), pWS(i,2), pWS(i,1), nm(i,1), bs(i,1), x_dev, Ctype);
    A(1:size(Alpha,1),i) = Alpha;
    B(1:size(Beta,1),i)  = Beta;
end

% Solve LDM
[x,fval,Aeq,beq,Aineq,bineq,f,lambda] = LDM_fun(A,B,col,W,Wmax,D,te,nm,bs,ns,np,F,Erem,BOini,dbug);
```

case_I_post.m

```
dec = 3;
```

```
xs = reshape(x,ns,size(x,1)/ns);
xt = xs(r,:);

out_X = zeros(1,6);

if np == 1
    if strcmp(Ttype, 'normal')
        out_G  = round_array(xt(1,5),dec);
        out_X(1,1:3)  = round_array(xt(1,6:8),dec);
        out_E  = round_array(xt(1,9),dec);
    elseif strcmp(Ttype, 'floor')
        out_G  = floor(xt(1,5));
        out_X(1,1:3)  = floor(xt(1,6:8));
        out_E  = floor(xt(1,9));
    elseif strcmp(Ttype, 'ceil')
        out_G  = ceil(xt(1,5));
        out_X(1,1:3)  = ceil(xt(1,6:8));
        out_E  = ceil(xt(1,9));
    elseif strcmp(Ttype, 'round')
        out_G  = round(xt(1,5));
        out_X(1,1:3)  = round(xt(1,6:8));
        out_E  = round(xt(1,9));
    end

    out_W  = [round_array(xt(1,1:3),dec),zeros(1,3)];
    out_W0 = round_array(W',dec);
    out_B0 = round_array(xt(1,10),dec);
    out_C = round_array(te',dec);
    for i=1:6
        out_C(i)=sum(out_C(i:3));
    end
end
```

LDM_fun.m

```
function [x,fval,Aeq,beq,Aineq,bineq,f,lambda]
= LDM_fun(A,B,col,W,Wmax,D,te,nm,bs,ns,np, F, Erem, BOini, dbug)

ss  =    12;

ci  = (3*F(1,1) + 1 )*F(1,2);                  % FGI holding cost          [-]
cfp = (te./60)./(nm.*ss);                      % Number of shifts per lot 1/tau(!) [ 1/hr ]
cfs = [ 1/Wmax(1,1); 1/Wmax(1,2); 1/Wmax(1,3); 1/Wmax(1,2); 1/Wmax(1,1); 1/Wmax(1,3)];

n = 8*ns;
m = 16*ns;

for i=1:3, sa(i) = size(B(1:max(find(B(:,i))),i),1);,end

W = sparse(reshape(W,size(W,1)/np,np));
D = sparse(reshape(D,size(D,1)/np,np));

% cost definition
f = LDM_fun_f(ns, np, F(1,3), F(1,5), F(1,2), F(1,4), ci, F(1,6), F(1,1));

% >>> equality constraints <<<
Aeq = LDM_fun_Aeq(ns, np, bs(1));
beq = LDM_fun_beq(ns, np, W, D, Erem, BOini);
```

```
% >>> inequality constraints <<<
Aineq = LDM_fun_Aineq(A,B,col,ns,np,sa,cfp,cfs);
bineq = LDM_fun_bineq(A,B,W,col,ns,np,sa);

% >>> lower and upper bounds <<<
ub=[];
for i=1:3, ub = [ub; (1/cfs(i))*ones(ns,1)]; end; ub = [ub; Inf*ones(2*ns,1)];
for i=1:3, ub = [ub; (1/cfp(i))*ones(ns,1)]; end; ub = [ub; Inf*ones(2*ns,1)];

lb=sparse(np*10*ns,1);
for i = 1 : np, lb((i-1)*m + 1 : (i-1)*m + ns,1) = 0.5*(bs(1)-1)*ones(ns,1); end

if dbug == 1, options = optimset('Display','iter','MaxIter',200); else options = []; end

[x,fval,exitflag,output,lambda] = linprog(f,Aineq,bineq,Aeq,beq,lb,ub,[],options);
```

LDM_fun_Aeq.m

```
function [Aeq] = LDM_fun_Aeq(ns, np, bs)

n    =    5*ns;
m    =   10*ns;
Aeq  = sparse(n*np,m*np);

Rw   = -1*ones(ns,1);Rw(ns,1)=0;
AeqW = spdiags( repmat(Rw,4*ns,1),-1,4*ns,4*ns) + spdiags(ones(4*ns,1),0,4*ns,4*ns);
AeqX = spdiags(-1*ones(4*ns,1),0,4*ns,4*ns+ns) + spdiags(ones(4*ns,1),ns,4*ns,4*ns+ns);

for j=1:ns, AeqX(:, ns + j)= bs.*AeqX(:, ns + j); end

AeqU = [AeqW,AeqX, sparse(4*ns,ns)];
AeqL = [sparse(ns,8*ns), spdiags(ones(ns,1),0,ns,ns), spdiags(ones(ns,1),0,ns,ns)+spdiags(Rw,-1,ns,ns)];
Aeqs = [ AeqU; AeqL ];

for i = 0: np-1, Aeq(i*n+1:(i+1)*n, i*m+1: (i+1)*m) = Aeqs; end
```

LDM_fun_beq.m

```
function [beq] = LDM_fun_beq(ns, np, W, D, Erem, BOini)

beq = [];
for i = 1: np
    beqs = [];
    beqs = sparse(4*ns,1);
    for j = 1:4
        beqs((j-1)*ns+1,1) = W(j,i);
    end
    D(1,i) = D(1,i) + Erem(1,i) + BOini(1,i);
    beq = [beq; beqs; D(:,i)];
end
```

LDM_fun_Aineq.m

```
function [Aineq] = LDM_fun_Aineq(A,B,col,ns,np,sa,cfp,cfs)

AineqWS = [];
```

```
% >>> Aineq Upper part
for z = 1:3 %WS
    carW = sparse(ns*sa(z),ns);
    carX = sparse(ns*sa(z),ns);
    for j=1:ns
        carX((j-1)*sa(z)+1:j*sa(z),j) = ones(sa(z),1);
        if j < ns, carW((j-1)*sa(z)+1:(j+1)*sa(z),j) = ones(2*sa(z),1);
        else, carW((j-1)*sa(z)+1:j*sa(z),j) = ones(sa(z),1);
        end
    end
    carW = carW - carX;
    WWS = (carW'*spdiags(repmat(-A(1:sa(z),z),ns,1),0,ns*sa(z),ns*sa(z)))';
    XWS = carX;
    if z == 1
        CWSs = [ 0*ns+1, 1*ns,  5*ns+1,  6*ns];
        CWSs = [ CWSs; CWSs + 9*ns*ones(1,4); CWSs + 18*ns*ones(1,4)];
        AineqWSs = LDM_fun_AineqRa(col, WWS, XWS, repmat([cfp(1),cfp(5)],[1,np]), ns*sa(z), 10*ns*np, CWSs);
    elseif z == 2
        CWSs = [ 1*ns+1, 2*ns,  6*ns+1, 7*ns];
        CWSs = [ CWSs; CWSs + 9*ns*ones(1,4); CWSs + 18*ns*ones(1,4)];
        AineqWSs = LDM_fun_AineqRa(col, WWS, XWS, repmat([cfp(2),cfp(4)],[1,np]), ns*sa(z), 10*ns*np, CWSs);
    elseif z == 3
        CWSs = [ 2*ns+1, 3*ns, 7*ns+1, 8*ns];
        CWSs = [ CWSs; CWSs + 9*ns*ones(1,4); CWSs + 18*ns*ones(1,4)];
        AineqWSs = LDM_fun_AineqRa(col, WWS, XWS, repmat([cfp(3),cfp(6)],[1,np]), ns*sa(z), 10*ns*np, CWSs);
    end
    AineqWS = [AineqWS; AineqWSs];
end

% >>> Aineq lower part
Rw=[];

AineqH = [spdiags( ones(3*ns,1),0,3*ns,3*ns )];
for i = 1:3, Rw = [Rw;repmat(cfs(i,1),ns,1)];,end
Qw = spdiags(sparse([Rw;repmat(0,ns,1)]),0,3*ns,3*ns);

AineqW=[];
for i = 1:np, AineqW = [AineqW, AineqH*Qw, sparse(3*ns,ns), sparse(3*ns,6*ns)]; end

Aineq = [ AineqWS; AineqW];

%============================ NEW FUNCTION ============================
%                       First recursive function
function [AineqR] = LDM_fun_AineqRa(col,WWS,XWS,cfp,n,m,C)

AineqR=[];
while size(col,1) > 1
    AineqR = [AineqR; LDM_fun_AineqRb(col,WWS,XWS,cfp,n,m,C)]; col = col(2:size(col,1),:);
end
AineqR = [AineqR; LDM_fun_AineqRb(col,WWS,XWS,cfp,n,m,C)];

%============================ NEW FUNCTION ============================
%                      Second recursive function
function [AineqR] = LDM_fun_AineqRb(col,WWS,XWS,cfp,n,m,C)

AineqR=sparse(n,m);
for j=1:size(col,2)
    AineqR(1:n, C(j,1):C(j,2)) = col(1,j).*WWS;
    AineqR(1:n, C(j,3):C(j,4)) = col(1,j).*cfp(j).*XWS;
end
```

LDM_fun_bineq.m

```
function [bineq] = LDM_fun_bineq(A,B,W,col,ns,np,sa)

bineqWS  = [];

for z = 1:3 %ws
    Wp=[];
    for j=1:np
        if z == 1
            Wp = [Wp;W(1,j)];%;W(5,j)];
        elseif z == 2
            Wp = [Wp;W(2,j)];%;W(4,j)];
        elseif z == 3
            Wp = [Wp;W(3,j)];%;W(6,j)];
        end
    end

    bineqWSs = LDM_fun_bineqRa(col, [A(1:max(find(B(:,z))),z)
              ; repmat(zeros(max(find(B(:,z))),1),ns-1,1)]
              , repmat(B(1:max(find(B(:,z))),z),ns,1),Wp , ns*sa(z));
    bineqWS = [bineqWS; bineqWSs];
end
bineq = [ bineqWS; ones(3*ns,1)];
bineq = sparse(bineq);

%========================== NEW FUNCTION ==========================
%                        First recursive function
function [bineqR] = LDM_fun_bineqRa(col,AWS,BWS,W,n)

bineqR=[];

while size(col,1) > 1
    bineqR = [bineqR; LDM_fun_bineqRb(col,AWS,BWS,W,n)]; col = col(2:size(col,1),:);
end
bineqR = [bineqR; LDM_fun_bineqRb(col,AWS,BWS,W,n)];

%========================== NEW FUNCTION ==========================
%                       Second recursive function
function [bineqR] = LDM_fun_bineqRb(col,AWS,BWS,W,n)

bineqR(1:n, 1) = BWS + (col(1,:)*W)*AWS;
```

LDM_fun_f.m

```
function [f] = LDM_fun_f(ns, np, cr, cd, cw, cp, ci, cb, scw)

fs=[];

for i=1:3, fs = [fs; cw * (1 + (i-1)*scw)*ones(ns,1)]; end

fs= [ fs; ci*ones(ns,1); cr*ones(ns,1); cp*ones(3*ns,1); cd*ones(ns,1); cb*ones(ns,1)];

f = [];
for j = 1 : np, f = [f; ((np+1-j)/np)*fs];end
```

ECF_fun_capcon.m

```
function [A,B] = ECF_fun_capcon(ca2, ce2, te, m, k, x_dev, Ctype)

alpha_tol  = 1.0e-4;
x_tol      = 1.0e-2;

x          = 0;
n          = 1;
i          = 1;

ck2        = (ca2/k + ce2)/2;
delta_m    = m * k / te;

if strcmp(Ctype, 'MRP')
    lin(n,4) = delta_m;

elseif strcmp(Ctype, 'MRP-C') | (strcmp(Ctype, 'CF') & ck2==0)
    lin(n,:) = [ ECF_fun_funcx(x,ck2,m,k,te) x ECF_fun_dfuncx(x,ck2,m,k,te)^-1 0 0];
    lin(n,4) = lin(n,2) - lin(n,3) * lin(n,1);
    lin(n+1,4) = delta_m;

elseif strcmp(Ctype, 'CF') & ck2 > 0
    lin(n,:) = [ ECF_fun_funcx(x,ck2,m,k,te) x ECF_fun_dfuncx(x,ck2,m,k,te)^-1 0 0];
    lin(n,4) = lin(n,2) - lin(n,3) * lin(n,1);

    while lin(n,3) > alpha_tol                  % while the slope of the tangent > tolerance
        xlower = lin(n,2);
        xupper = delta_m;
        xmid = (xlower + xupper)/2;
        x = xmid;

        while (xupper - xlower > x_tol),        % while the width of the observation range > tolerance
            fmid = ECF_fun_funcx(x,ck2,m,k,te);
            gmid = lin(n,3) * fmid + lin(n,4);

            if ( gmid / xmid - x_dev <= 0 ), xlower = xmid; else xupper = xmid; end

            xmid = (xlower + xupper)/2;
            x = xmid;
            i = i + 1;
        end
        n = n + 1;
        lin(n,:) = [ ECF_fun_funcx(x,ck2,m,k,te) x ECF_fun_dfuncx(x,ck2,m,k,te)^-1 0 i];
        lin(n,4) = lin(n,2) - lin(n,3) * lin(n,1);
        i = 1;
    end
    lin(n+1,4) = delta_m;
end

A = lin(:,3)./max(lin(:,4));
B = lin(:,4)./max(lin(:,4));

%========================== NEW FUNCTION ==========================
%                       effective clearing function
function fx = ECF_fun_funcx(x,ck2,m,k,te)

gamma   = sqrt(2*(m+1));
epsilon = te/(m*k);

fx = (k-1)/2 + ck2 * k * epsilon^gamma * x^gamma/(1 - epsilon * x) + te * x;
```

```
%========================== NEW FUNCTION ==========================
%             derivative of the effective clearing function
function dfx = ECF_fun_dfuncx(x,ck2,m,k,te)

gamma   = sqrt(2*(m+1));
epsilon = te/(m*k);

dfx = ck2 * k * (epsilon)^gamma * (((gamma*x^(gamma-1))/(1 - epsilon * x))
      + ( epsilon*x^gamma/(1-epsilon * x)^2)) + te;
```

WS_par.m

```
function [out] = WS_par(Xa, Xe, est, Atype, Ahor, Aalpha, Etype)

if size(Xa,1) < size(Xa,2), Xa = Xa';, end

if strcmp(Etype, 'on') & size(Xa,1) > 1
    [ta, ca2] = average_calc(Xa, Atype, Ahor, Aalpha);
else ca2 = est(1,3);
end

if size(Xe,1) < size(Xe,2), Xe = Xe';, end

if strcmp(Etype, 'on') & size(Xe,1) > 1
    [te, ce2] = average_calc(Xe, Atype, Ahor, Aalpha);
else te = est(1,1); ce2 = est(1,2);
end

out = [te,ce2,ca2];
```

demand_gen.m

```
function [Ds] = demand_gen(p, ns, Dtype, Dini, Dper, Dmin, Dmax)

if strcmp(Dtype, 'stepup')
    Ds = Dmax*ones(ns,1);
    if p <= Dini, Ds(1:min(Dini-p+1,ns), 1) = Dmin*ones(min(Dini-p+1,ns), 1);, end

elseif strcmp(Dtype, 'stepdown')
    Ds = Dmin*ones(ns,1);
    if p <= Dini, Ds(1:min(Dini-p+1,ns), 1) = Dmax*ones(min(Dini-p+1,ns), 1);, end

elseif strcmp(Dtype, 'rampup')
    Ds = Dmax*ones(ns,1);
    for i = 1:Dper, T(i,1) = Dmin + i/Dper*(Dmax-Dmin);, end
    F = [Dmin*ones(Dini,1); T; Ds];
    if p < (Dini + Dper), Ds(1:min((Dini+Dper)-p,ns), 1) = F(p: p + min((Dini+Dper)-p,ns)-1, 1);, end

elseif strcmp(Dtype, 'rampdown')
    Ds = Dmin*ones(ns,1);
    for i = 1:Dper, T(i,1) = Dmax + i/Dper*(Dmin-Dmax);, end
    F = [Dmax*ones(Dini,1); T; Ds];
    if p < (Dini + Dper), Ds(1:min((Dini+Dper)-p,ns), 1) = F(p: p + min((Dini+Dper)-p,ns)-1, 1);, end

elseif strcmp(Dtype, 'sinus')
    A = (Dmax - Dmin)/2;
    B = (2*pi/Dper);
    C = -Dini+max(p-Dini,0)+1.75*Dper;
```

```
    D = mean([Dmin,Dmax]);
    j=Dini;
    for i = 1:ns
        if i <= (Dini-p)
            Ds(i,1) = Dmin;
        else Ds(i,1) = A*sin(B*(j+C)) + D; j=j+1;
        end
    end
end
```

average_calc.m

```
function [avg, csq] = average_calc(X, Atype, Ahor, Aalpha)

if strcmp(Atype, 'average')                                % 'Normal' average function
    n   = length(X)
    avg = sum(X)/n;
    var = sum( (X - avg*ones(n,1)).^2)/(n-1);
    csq = var / avg^2;

elseif strcmp(Atype, 'n-average')                          % Average based on last n realizations
    n   = length(X);
    m   = min( n, Ahor);
    avg = sum(X(1:m))/m;
    var = sum((X(1:m) - avg*ones(m,1)).^2)/(m-1);
    csq = var / avg^2;

elseif strcmp(Atype, 'ewma')                               % Exponetial Weighted Moving Average function
    Xf(1,1) = X(1,1);
    for i = 2: length(X)
        Xf(i,1) = Aalpha*X(i,1) + (1- Aalpha)*Xf(i-1,1);
    end
    n   = length(Xf);
    avg = sum(Xf)/n;
    var = sum( (Xf - avg*ones(n,1)).^2)/(n-1);
    csq = var / avg^2;
end
```

round_array.m

```
function [R] = round_array(array, dec)
R = round(10^dec.*array)./10^dec;
```

filter_W.m

```
function [Wfil] = filter_W(W, Wmax)

if size(W,1) < size(W,2), W = W';,end

if W(1)+W(5) > Wmax(1)                  % Workstation 1
    pW(1) = W(1)/(W(1)+W(5));
    Wfil(1,1) = round(pW(1)*Wmax(1));
    Wfil(5,1) = Wmax(1) - Wfil(1);
else Wfil(1,1) = W(1); Wfil(5,1) = W(5);
end
```

```
if W(2)+W(4) > Wmax(2)                  % Workstation 2
    pW(2) = W(2)/(W(2)+W(4));
    Wfil(2,1) = round(pW(2)*Wmax(2));
    Wfil(4,1) = Wmax(2) - Wfil(2);
else Wfil(2,1) = W(2); Wfil(4,1) = W(4);
end

if W(3)+W(6) > Wmax(3)                  % Workstation 3
    pW(3) = W(3)/(W(3)+W(6));
    Wfil(3,1) = round(pW(3)*Wmax(3));
    Wfil(6,1) = Wmax(3) - Wfil(3);
else Wfil(3,1) = W(3); Wfil(6,1) = W(6);
end

Wfil(7) = W(7);
```

C.2 χ model

As mentioned in Appendix B, the heart of the simulation framework is the χ model. It encompasses the actual Discrete Event Simulation Model (DESM) as well as the distributed low level control layer.

The general form of the χ model is introduced in Chapter 4 and illustrated by Figure 4.5. A detailed representation (at system level) of the interaction (communication) between the different χ-processes is illustrated by Figure C.2. The interaction between the χ-processes, at workstation level, is illustrated by Figure C.3.

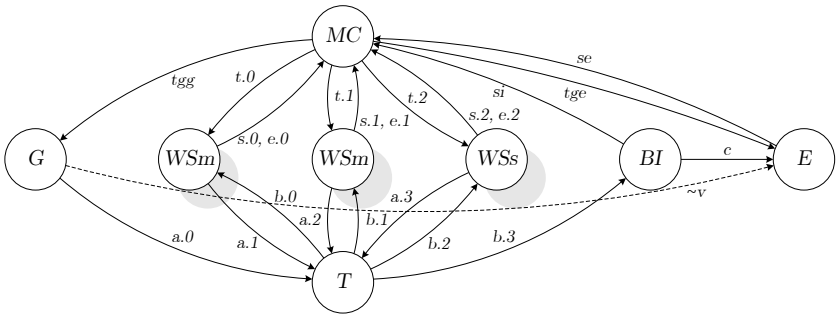
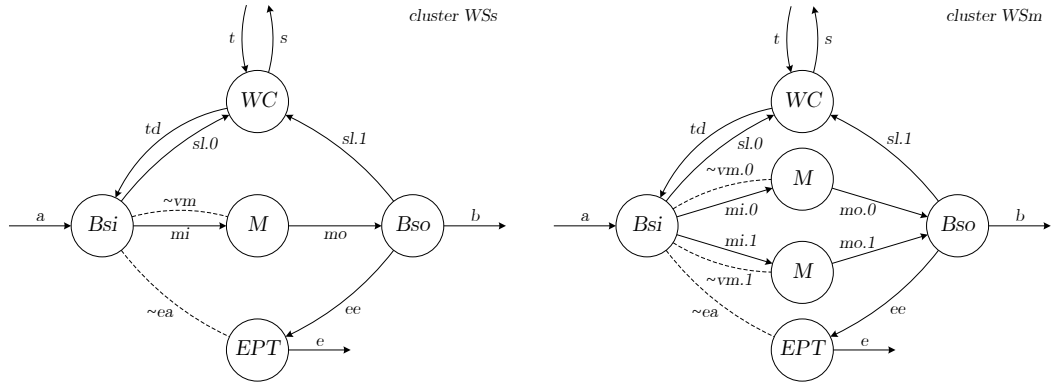


Figure C.2: Communication between the different χ processes, at system level

χ code

The χ-code describing the individual processes and functions of the DESM is presented in the remainder of this section. A general description of the individual processes is presented in Chapter 4.



(a) Single resource workstation (b) Multiple resource workstation

Figure C.3: Communication between the differen χ processes, at workstation level

Instantiation

```
from std import *
from random import *
from fileio import *
from io import *

type lot = iid.nat#itp.nat#isp.nat#irt.real#idd.real
   , trg = (real)^6
   , stat = (nat)^7

const LR : (nat)^4  = <|0,1,2,3|>          // lot routing
    , SS : real     = 720.0                // shift size [min]
    , NS : nat      = 6                    // number of steps
    , N  : nat      = 3                    // real number of steps
    , NP : nat      = 1                    // number of products
```

General purpose functions

```
// Determines the state of the buffer based on its contents
func f_stat(xs:lot*) -> stat =
|[ i:nat, st:stat
 | i:= 0
 ; *[ i < NS +1 -> st.i:= len([ x | x:lot <- xs, x.isp = i]); i:= i + 1 ]
 ; ret st
]|

// Updates step-id of a lot after processing
func f_uplot(x:lot, up: bool) -> lot =
|[ [ up -> x.isp:= x.isp + 1 | not up -> x.isp:= x.isp - 1 ]
   ; ret x
 ]|

// Determines processing time distribution based on issued parameters
func f_dist(te: (real^3)^6, tm: real) -> (->real)^6 =
|[ D:(->real)^6, i:nat, td,cd2:real
```

```
 | i:=0
 ; *[ i < NS and (te.i).1 /= 0.0
     -> td  := (1.0 - tm) * (te.i).0
      ; cd2 := ((((te.i).0)^2)*(te.i).1)/td^2
      ; D.i := gamma(1/cd2, cd2*td)
      ; i := i + 1
    | i < NS and (te.i).1  = 0.0
     -> D.i := constant(td)
      ; i := i + 1
    ]
 ; ret D
 ]|

// Update targets with batch send
func f_addstat(st1,st2: stat) -> stat =
|[ i: nat
 | i := 0; *[i < NS + 1 -> st1.i:= st1.i + st2.i; i:= i + 1 ]; ret st1
]|
```

Generator

```
// Release lots into the production line if issued targets permit
proc G(a: !lot*, t: ?real^2, v: !void)=
|[ m,n, n_ini: nat, ta,tg,dd: real, tgn: real^2
 | n_ini := GetNat("n_ini")
 ; ta := GetReal("ta")
 ; m := 0
 ; n := n_ini + 1
 ; tg := 0.0
 ; *[ tg >= 1.0; a! [< n, 0, 0, time, time + dd + m*ta>]
      -> v!; tg := tg - 1.0
       ; m := m + 1 ; n := n + 1
    | true; t? tgn
      -> tg := tgn.0
       ; dd := tgn.1
       ; m := 0
    ]
 ]|
```

Incoming buffers

```
// Increasing slack predicate for 'sort' function
func f_insl(x,y:lot#real) -> bool = |[ ret x.1 < y.1 ]|

// lot sequencing policy
func f_seqpol(xs:lot*, rs:nat*, tg,zt:trg, b:nat, t:real)-> lot*#nat =
|[ ys, zs: lot*
 | ys := []; zs:=[]
 ; ys:=  [ y.0
         | y: lot#real
           <- sort([<x, ((x.idd - zt.(x.isp) - t)/(N - x.isp))>
                | x:lot <- xs, (b*tg.(x.isp)) > 0.99 ]
                , f_insl
                   )
         ]
 ; *[ len(zs) < b and len(ys) > 0
     -> [ b*tg.((hd(ys)).isp) > 0  -> zs := zs ++ [hd(ys)]
                                    ; tg.((hd(ys)).isp) := tg.((hd(ys)).isp) - (1.0 / n2r(b))
```

```
        | b*tg.((hd(ys)).isp) <= 0 -> skip
        ]
        ; ys := tl(ys)
    ]
 ; ret< zs, hd(rs) >
 ]|

// Update issued targets
func f_uptrg(tg:trg, xs:lot*, b:nat) -> trg =
|[ i: nat, st: stat
 | i:=0; st:= f_stat(xs)
 ; *[i < NS -> tg.i:= tg.i - (n2r(st.i)/n2r(b)); i:= i + 1 ]; ret tg
]|

// Convert issued production targets
func f_data(rc:trg^2)-> trg^2=
|[ tg,zt:trg, i:nat
 | i := 0; tg := rc.0
 ; *[ i < NS -> zt.i := SS*(rc.1).i; i := i+1]; ret <|tg,zt|>
 ]|

// f_print is used to convert both the old and new targets into a string
func f_print(t:real, eqid:string ,tg,ac:trg ) -> string =
|[ i,j:nat, str:string
 | str:= r2s(t) ++ "\t " ++ eqid
 ; i:=0; *[ i < NS -> str := str ++ "\t " ++ r2s(tg.i); i:= i + 1 ]
 ; j:=0; *[ j < NS -> str := str ++ "\t " ++ r2s(ac.j); j:= j + 1 ]
 ; str := str ++ "\n"
 ; ret str
 ]|

// Incoming buffer for a workstation with a single resource
proc Bsi( a: ?lot*, b: !lot*#real
        , s: !string#lot*, t: ?trg^2
        , ea: !void
        , v: ?void, f,g: !file
        , wi: nat
        )=
|[ as:real*, xs,ys,zs: lot*, r: nat, rs: nat*
 , tg,tgo,zt: trg, rc: trg^2
 , eqid: string
 , bs: nat^3
 | ys:= GetWini("w_ini").wi
 ; rs:= []; as:= []
 ; bs := GetBs("bs")
 ; eqid:="Bsi"++n2s(wi)
 ; s! <"A",ys>
 ; t? rc
 ; <|tg,zt|>:= f_data(rc); tgo:= tg
 ; f! eqid, "\t", time, "\t", len(ys), "\n"
 ; *[ true
     -> [ len(rs) > 0 -> <zs,r> := f_seqpol(ys, rs, tg, zt, bs.wi, time)
        | len(rs) = 0 -> skip
        ]
      ; [ floor(len(ys)/(bs.wi)) >  len(as) -> as:= as ++ [time]
        | floor(len(ys)/(bs.wi)) <= len(as) -> skip
        ]
      ; [ true; a? xs
          -> ys:= ys ++ xs
           ; s! <"A",xs>; ea!
           ; f! eqid, "\t", time, "\t", len(ys), "\n"
        | len(zs) >= bs.wi; b! <zs, hd(as)>
```

```
          -> tg:= f_uptrg(tg,zs,bs.wi)
           ; ys:= ys -- zs
           ; rs:= rs -- [r]
           ; as := tl(as)
           ; f! eqid, "\t", time, "\t", len(ys), "\n"
        | true; t? rc
          -> g!f_print(time, eqid, tgo, tg)
           ; <|tg, zt|>:= f_data(rc)
           ; tgo := tg
        | true; v?
          -> rs:= rs ++ [0]
        ]
    ]
 ]|

// Incoming buffer for a workstation with a multiple resources
proc Bmi( a: ?lot*, b: (!lot*#real)^2
        , s: !(string#lot*), t: ?(trg^2)
        , ea: !void
        , v: (?void)^2, f,g: !file
        , wi: nat)=
|[ as: real*, xs,ys,zs: lot*, r: nat, rs: nat*, tg,tgo,zt: trg, rc: trg^2
 , eqid: string
 , bs: nat^3
 | ys:= GetWini("w_ini").wi
 ; rs:= []; as:=[]
 ; bs := GetBs("bs")
 ; eqid:= "Bmi"++n2s(wi)
 ; s! <"A",ys>
 ; t? rc
 ; <|tg,zt|>:= f_data(rc); tgo:= tg
 ; f! eqid, "\t", time, "\t", len(ys), "\n"
 ; *[ true
     -> [ len(rs) > 0 -> <zs,r> := f_seqpol(ys, rs, tg, zt, bs.wi, time)
        | len(rs) = 0 -> skip
        ]
      ; [ floor(len(ys)/(bs.wi)) >  len(as) -> as:= as ++ [time]
        | floor(len(ys)/(bs.wi)) <= len(as) -> skip
        ]
      ; [ true; a?xs
          -> ys:= ys ++ xs
           ; s! <"A",xs>
           ; ea!
           ; f! eqid, "\t", time, "\t", len(ys), "\n"
        | len(zs) >= bs.wi; b.r!<zs,hd(as)>
          -> tg := f_uptrg(tg, zs, bs.wi)
           ; ys := ys -- zs
           ; rs := rs -- [r]
           ; as := tl(as)
           ; f! eqid, "\t", time, "\t", len(ys), "\n"
        | true; t?rc
          -> g!f_print(time, eqid, tgo, tg)
           ; <|tg, zt|> := f_data(rc)
           ; tgo := tg
        | j: nat <- 0..2: true; v.j? -> rs:= rs ++ [j]
        ]
    ]
 ]|
```

Machine

```
// Determine need for setup-times
func f_setup(lt,pt,ls,ps,wi:nat) -> nat^3 =
|[ n:nat
 | [ wi = 2 -> [ lt  = pt and ls  = ps -> n:= 0
              | lt /= pt and ls  = ps -> n:= 1; pt:= lt
              | lt  = pt and ls /= ps -> n:= 2         ; ps:= ls
              | lt /= pt and ls /= ps -> n:= 3; pt:= lt; ps:= ls
              ]
   | wi /= 2 -> n:= 0
   ]
 ; ret <|n,pt,ps|>
]|

// Machine
proc M(a: ?lot*#real, b: !lot*#real, v: !void, g: !file, wi,mi: nat)=
|[ xs,ys : lot*, i,n,pt,ps: nat, t: real, eqid: string, D: (->real)^6
 , te: (real^3)^6
 , AA,tm: real
 , to: (real^2)^6
 , ts:  real^4
 | AA:=0.0
 ; eqid:= "WS"++n2s(wi)++"_M"++n2s(mi)
 ; te := GetTe("te")
 ; tm := GetReal("tm")
 ; to := GetTo("to")
 ; ts := GetTs("ts")
 ; D := f_dist(te,tm)
 ; *[ true
     -> v!; a?<xs,AA>
      // setup (if appropriate)
      ; <|n,pt,ps|> := f_setup(hd(xs).itp, pt, hd(xs).isp, ps, wi)
      ; t:= time; delta (ts.n)
      ; i:=1; ys:=xs
      ; *[ len(ys)>0 -> g! eqid,"_b",i,"\t", t, "\t", time, "\t setup_",n ,"\n"; ys:=tl(ys); i:=i+1]
      // load
      ; t:= time; delta (to.(hd(xs).isp)).0
      ; i:=1; ys:=xs
      ; *[ len(ys)>0 -> g! eqid,"_b",i,"\t", t, "\t", time, "\t load \n"; ys:=tl(ys); i:=i+1]
      // processing time
      ; t:= time; delta ( tm*te.(hd(xs).isp).0 + sample D.(hd(xs).isp))
      ; i:=1; ys:=xs
      ; *[ len(ys)>0
          -> g! eqid,"_b",i,"\t", t, "\t", time, "\t processing_step_", (hd(ys).isp)+1, "\n"
         ; ys:=tl(ys); i:=i+1]
      // unload
      ; t:= time; delta (to.(hd(xs).isp)).1
      ; i:=1; ys:=xs
      ; *[ len(ys)>0 -> g! eqid,"_b",i,"\t", t, "\t", time, "\t unload \n"; ys:=tl(ys); i:=i+1]
      // send updated batch
      ; b! <[f_uplot(x,true)| x:lot <- xs],AA>
   ]
 ]|
```

Outbound buffers

```
// Outbound buffer for a workstation with a single resource
proc Bso(a: ?lot*#real, b: !lot*, s: !string#lot*, ee:!nat#real#real) =
   |[ xs,ys: lot*, AA:real
```

```
    | ys:= []
    ; *[ true       ; a?<xs,AA>
        -> s! <"F",xs>
         ; ee!<0,AA,time>
         ; ys:= ys ++ xs
       | len(ys) > 0; b![hd(ys)]
        -> s! <"D",[hd(ys)]>
         ; ys:= tl(ys)
       ]
 ]|

// Outbound buffer for a workstation with a multiple resources
proc Bmo(a: (?lot*#real)^2, b: !lot*, s: !string#lot*, ee: !nat#real#real) =
 |[ xs,ys: lot*, AA:real
  | ys:= []
  ; *[ j: nat <- 0..2: true; a.j?<xs,AA>
      -> s!<"F",xs>
       ; ee!<j,AA,time>
       ; ys:= ys ++ xs
     | len(ys) > 0; b![hd(ys)]
      -> s!<"D",[hd(ys)]>
       ; ys:= tl(ys)
     ]
 ]|
```

EPT algorithm

```
// Determine the squared coefficient of variation
func f_sqv(t,ti,s2: real, i: nat) -> real#real#real#nat=
|[ i := i + 1
 ; [ i >  1 -> s2 := s2 * (i-2) / (i-1) + (ti-t)^2 / i
   | i <= 1 -> s2 := 0.0
   ]
 ; t := (i-1)/i*t + ti/i; ret <t, s2, s2/t^2,i>
 ]|

// EPT process
proc EPT(ea: ?void, ee: ?nat#real#real, se: !real*^2, f1,f2,g: !file, wi:nat)=
|[p,q: nat
 , mta,sa2,ca2,tsa,mte,se2,ce2,eptr: real, ras,res: real*
 , j:nat, AA,AD:real, ADold:real^2
 , eqid: string
 | p:=0; q:=0; ras:= []; res:= [ ]; tsa:=0.0
 ; ADold:=<|0.0,0.0|>
 ; eqid:= "WS"++n2s(wi)++"_M"
 ;*[ true; ea?
    -> ras := [time - tsa] ++ ras
     ; <mta,sa2,ca2,p> := f_sqv(mta, time - tsa, sa2,p)
     ; tsa := time
     ; f1! time,"\t mta:  \t",mta,"\t sa2: \t",sa2, "\t ca2: \t", ca2,"\n"
  | true; ee?<j,AA,AD>
     -> res  := [AD - max(AA, ADold.j)] ++ res
      ; eptr :=  AD - max(AA, ADold.j)
      ; g! eqid,j,"\t", max(AA, ADold.j) , "\t", AD, "\t EPT \n"
      ; <mte,se2,ce2,q> := f_sqv(mte, eptr, se2, q)
      ; f2! time,"\tmte:\t",mte,"\tse2:\t",se2, "\tce2:\t", ce2,"\n"
      ; ADold.j := AD
  | true; se! <|ras, res|> -> skip
  ]
 ]|
```

Inventory buffer

```
proc BI(a: ?lot*, b: !lot*, s: !stat, f:!file) =
|[ xs,ys: lot*, eqid: string
 | ys:= []
 ; eqid:= "BI"
 ; f! eqid, "\t", time, "\t", len(ys), "\n"
 ; *[ true       ; a?xs
     -> ys:= ys ++ xs
      ; f! eqid, "\t", time, "\t", len(ys), "\n"
    | len(ys) > 0; b![hd(ys)]
     -> ys:= tl(ys)
      ; f! eqid, "\t", time, "\t", len(ys), "\n"
    | true       ; s!f_stat(ys)
     -> skip
    ]
 ]|
```

Transporter

```
proc T(a: (?lot*)^4, b: (!lot*)^4)=
|[ xs: lot*
 | *[ j:nat <- 0..4: true; a.j? xs -> b.(LR.(hd(xs).isp))! xs ]
 ]|
```

Workstation controller

```
proc WC(s:!stat, t:?trg^2, sl: (?string#lot*)^2, td:!trg^2)=
|[ xs:lot*, tg: trg^2, dat: string#lot*
 | sl.0?dat; xs:= dat.1
 ; s!f_stat(xs); t?tg; td!tg
 ; *[ true; s!f_stat(xs)
     -> t?tg; td!tg
    | j: nat <- 0..2: true; sl.j?dat
     -> [ dat.0 = "A" -> xs := xs ++ dat.1
        | dat.0 = "F" -> xs := (xs ++ dat.1) -- [f_uplot(x,false)| x:lot <- dat.1]
        | dat.0 = "D" -> xs := xs -- dat.1
        ]
    ]
 ]|
```

Main controller

```
proc MC( sl: (?stat)^3, si: ?stat, se: ?real
       , tgg: !real^2, t: (!trg^2)^3, tge: !real
       , e: (?real*^2)^3
       , f: !file )=
|[ s,st: stat, eqid: string
 , rst:(real*^2)^3
 , b1,b2,b3,bf:bool^3, bi,be,bg,bt:bool
 , Th,p:nat
```

```
 , est: (real^3)^3
 , te : (real^3)^6
 , Ap: string#nat#real
 , Cp,Ep,Tp: string
 , Dp: string#nat#nat#nat#nat
 , Er: real
 , BO: real
 , Fc: real#nat#nat#nat#nat#nat
 , tg: real#trg#trg#trg#real#real
 , dbug : nat
 | te := GetTe("te")
 ; Ap := GetAp("Ap")
 ; Cp := GetString("Cp")
 ; Dp := GetDp("Dp")
 ; Ep := GetString("Ep")
 ; Fc := GetFc("Fc")
 ; Tp := GetString("Tp")
 ; tg := Gettg("tg_ini")
 ; dbug := GetNat("dbug")
 ; est:= GetTeEst("te_est")
 ; BO := tg.5
 ; Th := GetNat("T")
 ; p:=1
 ; bf:=<|false,false,false|>
 ; eqid:= "MC"
 ; bt := true
 ; *[ bt -> st:=<|0,0,0,0,0,0,0|>
            ; b1:= bf; b2:=bf; bi:=false; be:=false
            ; *[ k:nat <- 0..3: not b1.k; sl.k?s   -> st := f_addstat(st,s); b1.k:=true
               | l:nat <- 0..3: not b2.l; e.l?rst.l-> b2.l:=true
               |                not bi  ; si?s     -> st := f_addstat(st,s); bi:=true
               |        not be  ; se?Er    -> be := true
               ]
    ; f! eqid, "\t", time, "\t", tg.2, "\t", st, "\n"
    ; bt := LPsolve(Th, NP, p, st, rst, est, Ap, Cp, Dp, Ep, Tp, Er, BO, Fc, dbug)
    ; tg := LPpost (time, 1, p, st, Er, BO)
    ; BO:=tg.5
    ; b3:= bf; bg:=false; be:=false
    ; *[ k:nat <- 0..3: not b3.k; t.k!<|tg.1,tg.3|> -> b3.k:=true
       |                not bg;   tgg!<|tg.0,(tg.3).0|>           -> bg:=true
       |                not be;   tge!tg.4          -> be:=true
       ]
    ; delta SS
    ; p := p +1
    ]
 ]|
```

Exit

```
// Determine rolling average
func f_tavg(tim,ti,avgim: real, im: nat)-> real^2 =
 |[[ti = 0.0 -> ret <| ti,avgim |>
   |ti > 0.0 -> ret <| ti, avgim*(tim/ti)+im*(ti-tim)/ti |>
   ]
 ]|

// Exit process
proc E( a:?lot*, s:!real, t:?real
      , vg:?void,  f1,f2:!file
      )=
```

```
|[ xs: lot*, i,w: nat, tg,tgn,tim,avgtp,avgct,avgw: real
 , n_ini : nat
 | i:=0
 ; n_ini := GetNat("n_ini")
 ; w:= n_ini; tg:=0.0
 ; tim := 0.0; avgtp := 0.0; avgct := 0.0; avgw :=0.0
 ;*[ true -> f1! time,"\tE\t",avgtp,"\t",avgct,"\t",avgw,"\n"
   ; !time,"\t E \t",w,"\n"
   ; [ true     ; s!tg-> t?tgn
                      ; tg := tgn
     | tg >= 1.0; a? xs -> tg := tg - 1.0
                       ; w := w - 1
                       ; [ hd(xs).iid > n_ini -> i:= i + 1
                                       ; avgtp := i / time
                                       ; avgct := avgct*(i - 1)/i + (time - hd(xs).irt)/i
                                       ; <| tim,avgw |> := f_tavg(tim, time, avgw, w)
                         | hd(xs).iid <= n_ini -> skip
                         ]
                         ; f2! time, "\t", hd(xs).iid, "\t", hd(xs).isp, "\t", hd(xs).irt
                             , "\t", hd(xs).idd,  "\n"
     | true  ; vg?  -> w := w + 1
                     ;  <| tim,avgw |> := f_tavg(tim, time, avgw, w)                ]
   ]
 ]|
```

Cluster definition

```
clus WSs(a: ?lot*, b: !lot*, s: !stat, t: ?trg^2, e:!(real*)^2, wi:nat, file1,file2,file3:!file)=
|[ sl:(-string#lot*)^2, td:(-trg^2)
 , mi, mo: (-lot*#real)
 , vm: (-void)
 , ea: (-void), ee: (-nat#real#real)
 | WC(s, t, sl, td)
|| Bsi(a, mi, sl.0, td, ea, vm, fileout("output/B_level.txt"), fileout("output/B_targets.txt"), wi)
|| M(mi, mo, vm, fileout("output/gantt.txt"), wi, 0)
|| Bso(mo, b, sl.1, ee)
|| EPT(ea, ee, e, file1,file2,file3, wi)
]|

// CLUSTER FOR WORKSTATION WITH MULTIPLE RESOURCES

clus WSm(a: ?lot*, b: !lot*, s: !stat, t: ?trg^2, e:!(real*)^2, wi:nat, file1,file2,file3:!file)=
|[ sl:(-string#lot*)^2, td:(-trg^2)
 , mi, mo: (-lot*#real)^2
 , vm: (-void)^2
 , ea: (-void), ee: (-nat#real#real)
 | WC(s, t, sl, td)
|| Bmi(a, mi, sl.0, td, ea, vm, fileout("output/B_level.txt"), fileout("output/B_targets.txt"), wi )
|| j: nat <- 0..2: M(mi.j, mo.j, vm.j, fileout("output/gantt.txt"), wi, j)
|| Bmo(mo, b, sl.1, ee)
|| EPT(ea, ee, e, file1,file2,file3, wi)
]|

// CLUSTER OF THE TOTAL PRODUCTION LINE

clus I()=
|[ a,b: (-lot*)^4, c:(-lot*)
 , tgg: (-real^2), tge,se: (-real)
 , si:(-stat), s:(-stat)^3
 , t:(-trg^2)^3
```

```
 , e:(-real*^2)^3
 , v:(-void)
 | G (a.0, tgg, v)
|| MC(s, si, se, tgg, t, tge, e, fileout("output/wip_stat.txt"))
|| WSm( b.0, a.1, s.0, t.0, e.0, 0, fileout("output/EPT_WS0_a.txt")
      , fileout("output/EPT_WS0_e.txt"), fileout("output/EPT_gantt.txt"))
|| WSm( b.1, a.2, s.1, t.1, e.1, 1, fileout("output/EPT_WS1_a.txt")
      , fileout("output/EPT_WS1_e.txt"), fileout("output/EPT_gantt.txt"))
|| WSs( b.2, a.3, s.2, t.2, e.2, 2, fileout("output/EPT_WS2_a.txt")
      , fileout("output/EPT_WS2_e.txt"), fileout("output/EPT_gantt.txt"))
|| T (a, b)
|| BI(b.3, c, si, fileout("output/B_level.txt"))
|| E (c, se, tge, v, fileout("output/E_avg.txt"), fileout("output/E_lot.txt") )
]|

xper = |[I()]|
```

Modified χ code

Due to the convergence issue of the LDM, discussed in Chapter 5, the control framework is applied in a rolling horizon framework. Consequently, several modifications to the χ-code are required. These modifications are presented in the remainder of this section.

General purpose functions

```
func f_addtrg(trg1,trg2: trg) -> trg =                  // Add new targets to old
|[ i: nat
 | i:=0; *[i < NS -> trg1.i:= trg1.i + trg2.i; i:= i + 1 ]; ret trg1
]|
```

Generator

```
// Release lots into the production line if issued targets permit
proc G(a: !lot*, t: ?real^2, v: !void)=
|[ m,n, n_ini: nat, ta,tg,dd: real, tgn: real^2
 | n_ini := GetNat("n_ini")
 ; ta := GetReal("ta")
 ; m:=0
 ; n:= n_ini + 1
 ; tg:=0.0
 ; *[ tg >= 1.0; a! [< n, 0, 0, time, time + dd + m*ta>]
      -> v!; tg := tg - 1.0
       ; m := m + 1 ; n := n + 1
   | true; t? tgn
     -> tg := tg + tgn.0
      ; dd := tgn.1
      ; m := 0
   ]
 ]|
```

Incoming buffers

```
// Incoming buffer for a workstation with a single resource
proc Bsi( a: ?lot*, b: !lot*#real
        , s: !string#lot*, t: ?trg^2
        , ea: !void
        , v: ?void, f,g: !file
        , wi: nat
        )=
|[ as:real*, xs,ys,zs: lot*, r: nat, rs: nat*
 , tg,tgo,zt: trg, rc: trg^2
 , eqid: string
 , bs: nat^3
 | ys:= GetWini("w_ini").wi
 ; rs:= []; as:= []
 ; bs := GetBs("bs")
 ; eqid:="Bsi"++n2s(wi)
 ; s! <"A",ys>
 ; t? rc
 ; <|tg,zt|>:= f_data(rc); tgo:= tg
 ; f! eqid, "\t", time, "\t", len(ys), "\n"
 ; *[ true
     -> [ len(rs) > 0 -> <zs,r> := f_seqpol(ys, rs, tg, zt, bs.wi, time)
        | len(rs) = 0 -> skip
        ]
      ; [ floor(len(ys)/(bs.wi)) >  len(as) -> as:= as ++ [time]
        | floor(len(ys)/(bs.wi)) <= len(as) -> skip
        ]
      ; [ true; a? xs
          -> ys:= ys ++ xs
           ; s! <"A",xs>; ea!
           ; f! eqid, "\t", time, "\t", len(ys), "\n"
        | len(zs) >= bs.wi; b! <zs, hd(as)>
          -> tg:= f_uptrg(tg,zs,bs.wi)
           ; ys:= ys -- zs
           ; rs:= rs -- [r]
           ; as := tl(as)
           ; f! eqid, "\t", time, "\t", len(ys), "\n"
        | true; t? rc
          -> g!f_print(time, eqid, tgo, tg)
           ; rc.0 := f_addtrg(rc.0, tg)
           ; <|tg, zt|>:= f_data(rc)
           ; tgo := tg
        | true; v?
          -> rs:= rs ++ [0]
        ]
   ]
 ]|

// Incoming buffer for a workstation with a multiple resources
proc Bmi( a: ?lot*, b: (!lot*#real)^2
        , s: !(string#lot*), t: ?(trg^2)
        , ea: !void
        , v: (?void)^2, f,g: !file
        , wi: nat)=
|[ as: real*, xs,ys,zs: lot*, r: nat, rs: nat*, tg,tgo,zt: trg, rc: trg^2
 , eqid: string
 , bs: nat^3
 | ys:= GetWini("w_ini").wi
 ; rs:= []; as:=[]
 ; bs := GetBs("bs")
 ; eqid:= "Bmi"++n2s(wi)
```

```
 ; s! <"A",ys>
 ; t? rc
 ; <|tg,zt|>:= f_data(rc); tgo:= tg
 ; f! eqid, "\t", time, "\t", len(ys), "\n"
 ; *[ true
     -> [ len(rs) > 0 -> <zs,r> := f_seqpol(ys, rs, tg, zt, bs.wi, time)
        | len(rs) = 0 -> skip
        ]
      ; [ floor(len(ys)/(bs.wi)) >  len(as) -> as:= as ++ [time]
        | floor(len(ys)/(bs.wi)) <= len(as) -> skip
        ]
      ; [ true; a?xs
          -> ys:= ys ++ xs
           ; s! <"A",xs>
           ; ea!
           ; f! eqid, "\t", time, "\t", len(ys), "\n"
        | len(zs) >= bs.wi; b.r!<zs,hd(as)>
          -> tg := f_uptrg(tg, zs, bs.wi)
           ; ys := ys -- zs
           ; rs := rs -- [r]
           ; as := tl(as)
           ; f! eqid, "\t", time, "\t", len(ys), "\n"
        | true; t?rc
          -> g!f_print(time, eqid, tgo, tg)
           ; rc.0 := f_addtrg(rc.0, tg)
           ; rc.0 := f_addtrg(rc.0, tg)
           ; <|tg, zt|> := f_data(rc)
           ; tgo := tg
        | j: nat <- 0..2: true; v.j? -> rs:= rs ++ [j]
        ]
    ]
 ]|
```

Exit

```
// Exit process
proc E( a:?lot*, s:!real, t:?real
      , vg:?void,  f1,f2:!file
      )=
|[ xs: lot*, i,w: nat, tg,tgn,tim,avgtp,avgct,avgw: real
 , n_ini : nat
 | i:=0
 ; n_ini := GetNat("n_ini")
 ; w:= n_ini; tg:=0.0
 ; tim := 0.0; avgtp := 0.0; avgct := 0.0; avgw :=0.0
 ;*[ true -> f1! time,"\tE\t",avgtp,"\t",avgct,"\t",avgw,"\n"
   ; !time,"\t E \t",w,"\n"
   ; [ true     ; s!tg-> t?tgn
                       ; tg:= tg + tgn
     | tg >= 1.0; a? xs -> tg := tg - 1.0
                         ; w := w - 1
                         ; [ hd(xs).iid > n_ini -> i:= i + 1
                                                 ; avgtp := i / time
                                                 ; avgct := avgct*(i - 1)/i + (time - hd(xs).irt)/i
                                                 ; <| tim,avgw |> := f_tavg(tim, time, avgw, w)
                           | hd(xs).iid <= n_ini -> skip
                           ]
                           ; f2! time, "\t", hd(xs).iid, "\t", hd(xs).isp, "\t", hd(xs).irt
                               , "\t", hd(xs).idd,  "\n"
     | true  ; vg?  -> w := w + 1
```

```
                    ;  <| tim,avgw |> := f_tavg(tim, time, avgw, w)                ]
  ]
 ]|
```

Appendix D

Case II Description

In this appendix, the files used during the implementation of the second case are described. A detailed description of the second case is presented in Section 4.4.

This appendix starts with a description of the case dependent Matlab functions. The χ-model of the second case is discussed in Section C.2.

D.1 Matlab functions

In this section, the Matlab functions, initiating the (case dependent) Linear Discrete Model (LDM), are described. The structure of these files is similar to that of the first case and is illustrated by Figure C.1.

Matlab code

In this subsection, only the Matlab functions differing from those of the first case are discussed. Functions identical to those of the first case are described in Section C.1.

case_II_standalone.m

```
ns     =    630;             % planning horizon
np     =     1;              % number of products
p      =     1;              % shift id
r      =     1;

dbug   =     0;              % dbug value

Atype  =    'average';       % Type of average calculation function {average, n-average, ewma}
Ahor   =    10;              % Horizon of realizations the average is based on
Aalpha =     0.5;            % Exponential Weighted Moving Average parameter
```

```
Ctype  =    'CF';           % Resource capacity constraint type {MRP, MRP-C, CF}

Dtype  =    'rampup';       % Type of demand function {stepup, stepdown, rampup, rampdown, sinus}
Dini   =    50;             % minimum value of demand function
Dper   =    150;             % maximum value of demand function
Dmin   =     0;             % Initialization period
Dmax   =     3;             % (Transition) Period length

Etype  =    'off';          % measure online EPT {on, off}

Ttype  =    'normal';       % Resource capacity constraint type {normal, floor, ceil, round}

Erem   =     0;             % Remains of the Exit target
BOini  =     0;             % initial value of back-orders

% Cost definition
F(1,1) =     0.5;           % Increase in invertors holding cost
F(1,2) =    20;             % Inventory holding cost
F(1,3) =     0;             % Release cost
F(1,4) =     0;             % Production cost
F(1,5) =     0;             % Demand cost
F(1,6) = 10000;             % Backorder cost

W = [1; 0; 0; 0; 1; 0; 0];   % Initial WIP value

EaWS1=[];
EeWS1=[];
EaWS2=[];
EeWS2=[];
EaWS3=[];
EeWS3=[];

estWS1 = [ 300, 0.92, 2.46];
estWS2 = [  70, 0.52, 1.58];
estWS3 = [  55.0, 0.77, 1.75];

Wmax   = [ 18 12 12 ];
```

case_II_main.m

```
% >>> INITIALIZATION
load activeconstraints.mat                    % loading active constraints matrix

Wmax   = [ 18 12 12 ];

ss     = 12;                                  % shift size
sp     =  5;                                  % Number of significant parameters
sd     =  3;

x_dev  =  1.1;                                % Maximum allowed deviation of the former tangent
nm     = [2; 2; 1; 2; 2; 1];
bs     = [3; 1; 1; 1; 3; 1];

if np == 1
    col = col13(5:7,5:6);
    %col = col13(5:7,5:6);
elseif np == 2
    col = col213;
elseif np == 3;
    col = col123;
```

```
end

W  = filter_W(W, Wmax);
D  = demand_gen(p, ns, Dtype, Dini, Dper, Dmin, Dmax);              % demand

pWS(1,1:3) = WS_par(EaWS1, EeWS1, estWS1, Atype, Ahor, Aalpha, Etype);     % te, ce2, ca2
pWS(2,1:3) = WS_par(EaWS2, EeWS2, estWS2, Atype, Ahor, Aalpha, Etype);
pWS(3,1:3) = WS_par(EaWS3, EeWS3, estWS3, Atype, Ahor, Aalpha, Etype);
te = [ pWS(1,1); pWS(2,1); pWS(3,1); pWS(2,1); pWS(1,1); pWS(3,1)];

% >>> EXECUTION
cfp = (te./60)./(nm.*ss);

% Effective linear clearing functions
for i = 1 : 3                                       % Workstations
    [Alpha,Beta] = ECF_fun_capcon( pWS(i,3), pWS(i,2), pWS(i,1), nm(i,1), bs(i,1), x_dev, Ctype);
    A(1:size(Alpha,1),i) = Alpha;
    B(1:size(Beta,1),i)  = Beta;
end

% Solve LDM
[x,fval,Aeq,beq,Aineq,bineq,f,lambda] = LDM_fun(A,B,col,W,Wmax,D,te,nm,bs,ns,np,F,Erem,BOini,dbug);
```

case_II_post.m

```
dec = 3;

xs = reshape(x,ns,size(x,1)/ns);
xt = xs(r,:);

if np == 1
    if strcmp(Ttype, 'normal')
        out_G  = round_array(xt(1,8),dec);
        out_X  = round_array(xt(1,9:14),dec);
        out_E  = round_array(xt(1,15),dec);
    elseif strcmp(Ttype, 'floor')
        out_G  = floor(xt(1,8));
        out_X  = floor(xt(1,9:14));
        out_E  = floor(xt(1,15));
    elseif strcmp(Ttype, 'ceil')
        out_G  = ceil(xt(1,8));
        out_X  = ceil(xt(1,9:14));
        out_E  = ceil(xt(1,15));
    elseif strcmp(Ttype, 'round')
        out_G  = round(xt(1,8));
        out_X  = round(xt(1,9:14));
        out_E  = round(xt(1,15));
    end
    out_W  = round_array(xt(1,1: 6),dec);
    out_W0 = round_array(W',dec);
    out_B0 = round_array(xt(1,16),dec);
    out_C = round_array(te',dec);
    for i=1:6
        out_C(i)=sum(out_C(i:6));
    end
end
```

LDM_fun.m

```
function [x,fval,Aeq,beq,Aineq,bineq,f,lambda] = LDM_fun(A,B,col,W,Wmax,D,te,nm,bs,ns,np, F, Erem, BOini, dbug)

ss  =    12;

ci  = (6*F(1,1) + 1 )*F(1,2);                 % FGI holding cost          [-]
cfp = (te./60)./(nm.*ss);                     % Number of shifts per lot 1/tau(!) [ 1/hr ]
cfs = [ 1/Wmax(1,1); 1/Wmax(1,2); 1/Wmax(1,3); 1/Wmax(1,2); 1/Wmax(1,1); 1/Wmax(1,3)];

n =  8*ns;
m = 16*ns;

for i=1:3, sa(i) = size(B(1:max(find(B(:,i))),i),1);,end

W = sparse(reshape(W,size(W,1)/np,np));
D = sparse(reshape(D,size(D,1)/np,np));

% cost definition
f = LDM_fun_f(ns, np, F(1,3), F(1,5), F(1,2), F(1,4), ci, F(1,6), F(1,1));

% >>> equality constraints <<<
Aeq = LDM_fun_Aeq(ns, np, bs(1));
beq = LDM_fun_beq(ns, np, W, D, Erem, BOini);

% >>> inequality constraints <<<
Aineq = LDM_fun_Aineq(A,B,col,ns,np,sa,cfp,cfs);
bineq = LDM_fun_bineq(A,B,W,col,ns,np,sa);

% >>> lower and upper bounds <<<
lb=sparse(np*16*ns,1);
for i = 1 : np
    lb((i-1)*m        + 1 : (i-1)*m +   ns,1)=0.5*(bs(1)-1)*ones(ns,1);
    lb((i-1)*m + 4*ns + 1 : (i-1)*m + 5*ns,1)=0.5*(bs(1)-1)*ones(ns,1);
end

ub=[];
for i=1:6, ub = [ub; (1/cfs(i))*ones(ns,1)]; end; ub = [ub; Inf*ones(2*ns,1)];
for i=1:6, ub = [ub; (1/cfp(i))*ones(ns,1)]; end; ub = [ub; Inf*ones(2*ns,1)];

if dbug == 1
    options = optimset('Display','iter', 'Diagnostics','on', 'maxIter',Inf);
else options = [];
end

[x,fval,exitflag,output,lambda] = linprog(f,Aineq,bineq,Aeq,beq,lb,ub,[],options);
```

LDM_fun_Aeq.m

```
function [Aeq] = LDM_fun_Aeq(ns, np, bs)

n   =  8*ns;
m   = 16*ns;
Aeq = sparse(8*ns*np,16*ns*np);

Rw = -1*ones(ns,1);
Rw(ns,1)=0;
AeqW = spdiags( repmat(Rw,7*ns,1),-1,7*ns,7*ns) + spdiags(ones(7*ns,1),0,7*ns,7*ns);
AeqX = spdiags(-1*ones(7*ns,1),0,7*ns,7*ns+ns) + spdiags(ones(7*ns,1),ns,7*ns,7*ns+ns);
```

```
for j=1:ns
    AeqX(:, ns + j)= bs.*AeqX(:, ns + j);
    AeqX(:, 5*ns + j)= bs.*AeqX(:, 5*ns + j);
end

AeqU = [AeqW,AeqX, sparse(7*ns,ns)];
AeqL = [sparse(ns,14*ns), spdiags(ones(ns,1),0,ns,ns), spdiags(ones(ns,1),0,ns,ns)+spdiags(Rw,-1,ns,ns)];
Aeqs  = [ AeqU; AeqL ];

for i = 0: np-1
    Aeq(i*n+1:(i+1)*n, i*m+1: (i+1)*m) = Aeqs;
end
```

LDM_fun_beq.m

```
function [beq] = LDM_fun_beq(ns, np, W, D, Erem, BOini)

beq = [];
for i = 1: np
    beqs = [];
    beqs = sparse(7*ns,1);
    for j = 1:7
        beqs((j-1)*ns+1,1) = W(j,i);
    end
    D(1,i) = D(1,i) + Erem(1,i) + BOini(1,i);
    beq = [beq; beqs; D(:,i)];
end
```

LDM_fun_Aineq.m

```
function [Aineq] = LDM_fun_Aineq(A,B,col,ns,np,sa,cfp,cfs)

AineqWS = [];
% >>> Aineq Upper part
for z = 1:3 %WS
    carW = sparse(ns*sa(z),ns);
    carX = sparse(ns*sa(z),ns);
    for j=1:ns
        carX((j-1)*sa(z)+1:j*sa(z),j) = ones(sa(z),1);
        if j < ns, carW((j-1)*sa(z)+1:(j+1)*sa(z),j) = ones(2*sa(z),1);
        else, carW((j-1)*sa(z)+1:j*sa(z),j) = ones(sa(z),1);
        end
    end
    carW = carW - carX;
    WWS = (carW'*spdiags(repmat(-A(1:sa(z),z),ns,1),0,ns*sa(z),ns*sa(z)))';
    XWS = carX;
    if z == 1
        CWSs = [ 0*ns+1, 1*ns,  8*ns+1,  9*ns; 4*ns+1, 5*ns, 12*ns+1, 13*ns];
        CWSs = [ CWSs; CWSs + 15*ns*ones(2,4); CWSs + 30*ns*ones(2,4)];
        AineqWSs = LDM_fun_AineqRa(col, WWS, XWS, repmat([cfp(1),cfp(5)],[1,np]), ns*sa(z), 16*ns*np, CWSs);
    elseif z == 2
        CWSs = [ 1*ns+1, 2*ns,  9*ns+1, 10*ns; 3*ns+1, 4*ns, 11*ns+1, 12*ns];
        CWSs = [ CWSs; CWSs + 15*ns*ones(2,4); CWSs + 30*ns*ones(2,4)];
        AineqWSs = LDM_fun_AineqRa(col, WWS, XWS, repmat([cfp(2),cfp(4)],[1,np]), ns*sa(z), 16*ns*np, CWSs);
    elseif z == 3
        CWSs = [ 2*ns+1, 3*ns, 10*ns+1, 11*ns; 5*ns+1, 6*ns, 13*ns+1, 14*ns];
        CWSs = [ CWSs; CWSs + 15*ns*ones(2,4); CWSs + 30*ns*ones(2,4)];
        AineqWSs = LDM_fun_AineqRa(col, WWS, XWS, repmat([cfp(3),cfp(6)],[1,np]), ns*sa(z), 16*ns*np, CWSs);
```

```
    end
    AineqWS = [AineqWS; AineqWSs];
end

% >>> Aineq lower part
Rw=[];

AineqH = [ spdiags( ones(3*ns,1),0,3*ns,3*ns )
         , spdiags( ones(ns,1),-ns,3*ns,ns )
         , spdiags( ones(ns,1),0 ,3*ns,ns )
         , spdiags( ones(ns,1),-2*ns ,3*ns,ns ),sparse(3*ns,ns)];
for i = 1:6, Rw = [Rw;repmat(cfs(i,1),ns,1)];,end
Qw = spdiags(sparse([Rw;repmat(0,ns,1)]),0,7*ns,7*ns);

AineqW=[];
for i = 1:np, AineqW = [AineqW, AineqH*Qw, sparse(3*ns,ns), sparse(3*ns,8*ns)]; end

Aineq = [ AineqWS; AineqW];

%========================== NEW FUNCTION ==========================
%                       First recursive function
function [AineqR] = LDM_fun_AineqRa(col,WWS,XWS,cfp,n,m,C)

AineqR=[];
while size(col,1) > 1
    AineqR = [AineqR; LDM_fun_AineqRb(col,WWS,XWS,cfp,n,m,C)]; col = col(2:size(col,1),:);
end
AineqR = [AineqR; LDM_fun_AineqRb(col,WWS,XWS,cfp,n,m,C)];

%========================== NEW FUNCTION ==========================
%                     Second recursive function
function [AineqR] = LDM_fun_AineqRb(col,WWS,XWS,cfp,n,m,C)

AineqR=sparse(n,m);
for j=1:size(col,2)
    AineqR(1:n, C(j,1):C(j,2)) = col(1,j).*WWS;
    AineqR(1:n, C(j,3):C(j,4)) = col(1,j).*cfp(j).*XWS;
end
```

LDM_fun_bineq.m

```
function [bineq] = LDM_fun_bineq(A,B,W,col,ns,np,sa)

bineqWS  = [];

for z = 1:3 %ws
    Wp=[];
    for j=1:np
        if z == 1
            Wp = [Wp; W(1,j); W(5,j)];
        elseif z == 2
            Wp = [Wp; W(2,j); W(4,j)];
        elseif z == 3
            Wp = [Wp; W(3,j); W(6,j)];
        end
    end

    bineqWSs = LDM_fun_bineqRa(col, [A(1:max(find(B(:,z))),z)
                                    ; repmat(zeros(max(find(B(:,z))),1),ns-1,1)]
                               , repmat(B(1:max(find(B(:,z))),z),ns,1),Wp , ns*sa(z));
```

```
    bineqWS = [bineqWS; bineqWSs];
end
bineq = [ bineqWS; ones(3*ns,1)];
bineq = sparse(bineq);

%============================ NEW FUNCTION ============================
%                        First recursive function
function [bineqR] = LDM_fun_bineqRa(col,AWS,BWS,W,n)

bineqR=[];

while size(col,1) > 1
    bineqR = [bineqR; LDM_fun_bineqRb(col,AWS,BWS,W,n)]; col = col(2:size(col,1),:);
end
bineqR = [bineqR; LDM_fun_bineqRb(col,AWS,BWS,W,n)];

%============================ NEW FUNCTION ============================
%                       Second recursive function
function [bineqR] = LDM_fun_bineqRb(col,AWS,BWS,W,n)

bineqR(1:n, 1) = BWS + (col(1,:)*W)*AWS;
```

LDM_fun_f.m

```
function [f] = LDM_fun_f(ns, np, cr, cd, cw, cp, ci, cb, scw)

fs=[];

for i=1:6, fs = [fs; cw * (1 + (i-1)*scw)*ones(ns,1)]; end

fs= [ fs; ci*ones(ns,1); cr*ones(ns,1); cp*ones(6*ns,1); cd*ones(ns,1); cb*ones(ns,1)];

f = [];
for j = 1 : np
    f = [f; ((np+1-j)/np)*fs];
end
```

D.2 χ model

As mentioned in Section 4.4, the χ model of the second re-entrant case is identical to that of the first case, described in Appendix C.2. The only alteration required is that of the routing table (part of the instantiation), representing the new product flow.

Instantiation

```
const LR : (nat)^4  = <|0,1,2,3|>          // lot routing
    , SS : real     = 720.0                // shift size [min]
    , NS : nat      = 6                    // number of steps
    , N  : nat      = 3                    // real number of steps
    , NP : nat      = 1                    // number of products
```

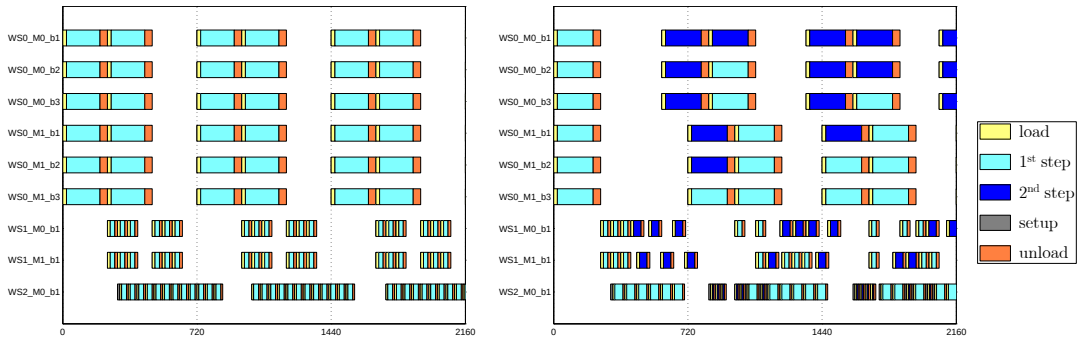
Appendix E

Experiments

In Chapter 5, the performance of the presented MPC framework is evaluated for the two cases described in Chapter 4. The results of the simulation experiments, defined in Section 5.1, are presented in this appendix.

E.1 Validation

Before the performance of the MPC framework can be discussed, the different parts of the control and simulation framework are validated. Validation of the DESM is achieved by comparing the output of a simulation experiment to manual calculations. To obtain a better insight, the state of the system is visualized using (production) Gantt charts.



(a) Case I (b) Case II

Figure E.1: Production Gantt chart

The Gantt charts of Figure E.1 are examples of the Gantt charts used for the validation of the DESM. Note that, since the manual derived Gantt chart and the (simulation) Gantt chart are identical, only a single Gantt chart per case is presented.

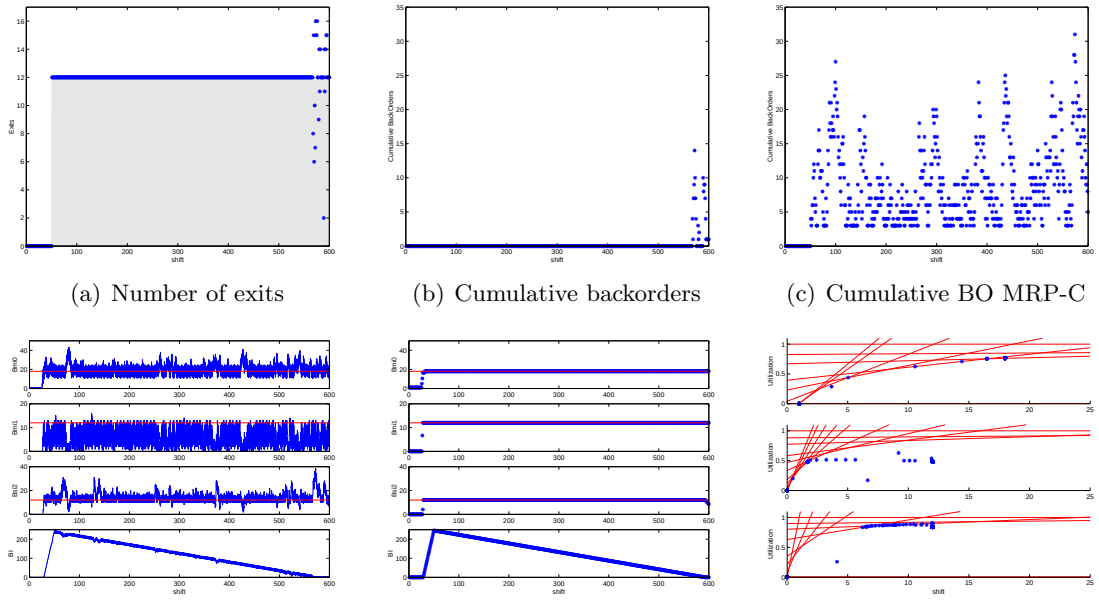
E.2 Simulation results Case I

In Section 5.3, the simulation results of the first case are discussed. The accompanying results are illustrated in this section. The simulation parameters of the conducted experiments are summarized in Table E.1.

	D_{ini}	D_{per}	D_{min}	D_{max}	sc_i	Run length	Planning horizon
1-a	50	0	0	12	(18, 12, 12)	600	630
1-b	50	0	0	12	(36, 24, 24)	600	630
2	50	150	0	12	(36, 24, 24)	600	630
3	48	96	0	12	(36, 24, 24)	600	630

Table E.1: Simulation parameters for the experiments of Case I

Experiment 1-a: Step up with original storage capacity

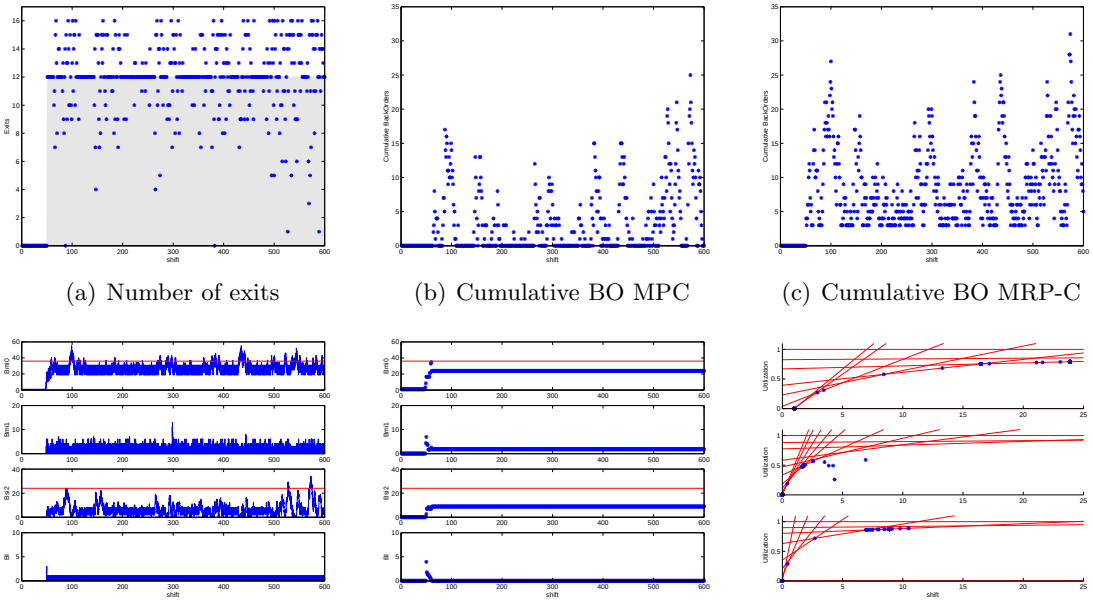


(a) Number of exits (b) Cumulative backorders (c) Cumulative BO MRP-C

(d) Buffer level (e) Predicted buffer level (f) Resource constraints

Figure E.2: Typical simulation results for the 'step up' experiment

Experiment 1-b: Step up with extended storage capacity

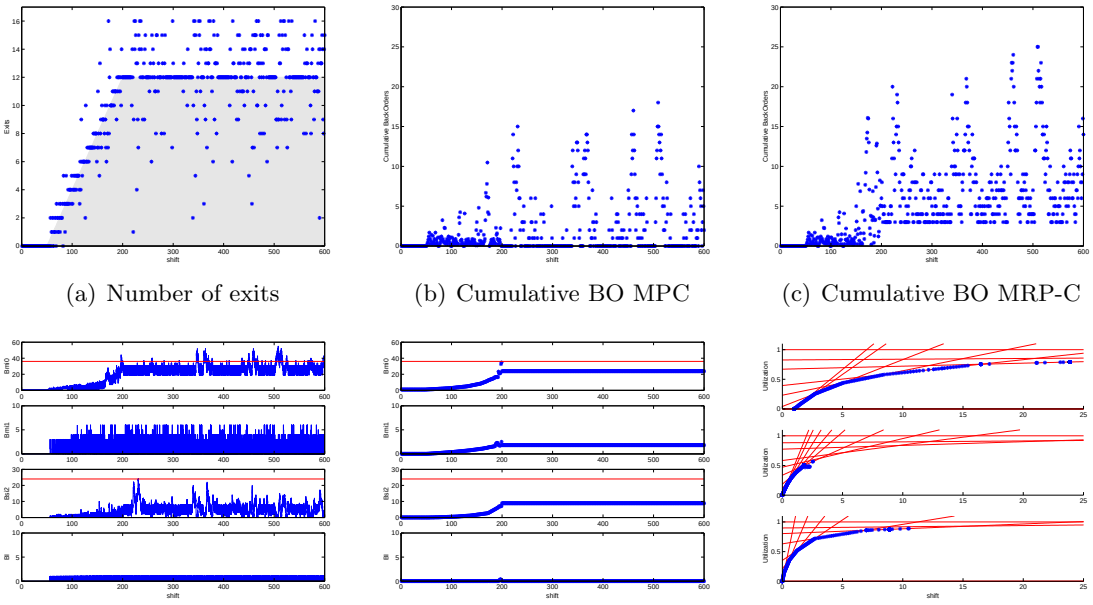


(a) Number of exits (b) Cumulative BO MPC (c) Cumulative BO MRP-C

(d) Buffer level (e) Predicted buffer level (f) Resource constraints

Figure E.3: Typical simulation results for the 'step up' experiment

Experiment 2: Ramp up with extended storage capacity

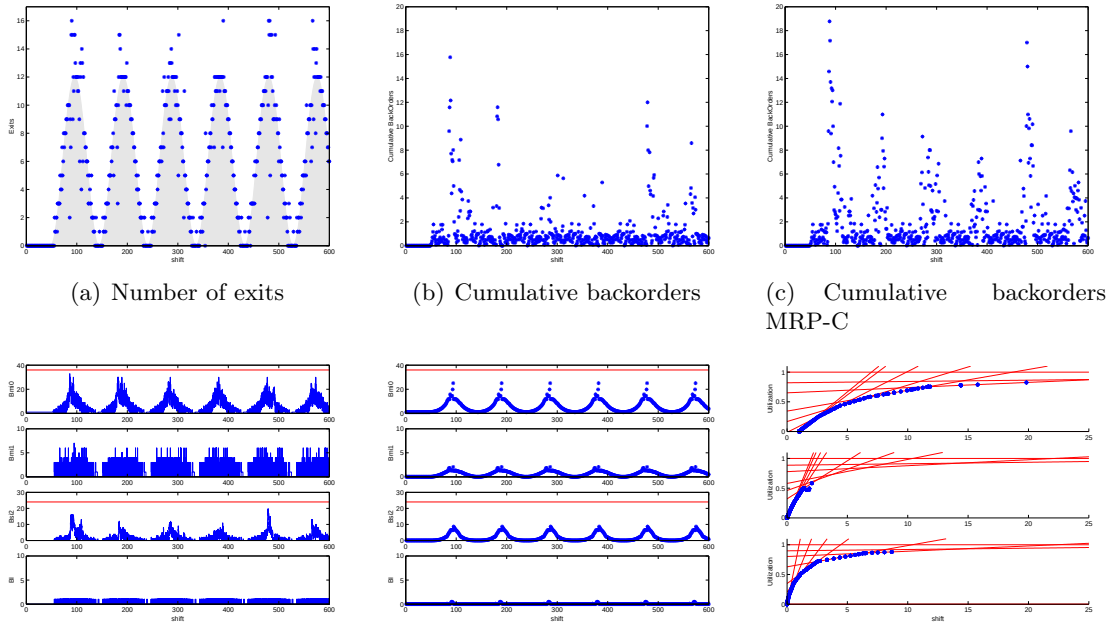


(a) Number of exits (b) Cumulative BO MPC (c) Cumulative BO MRP-C

(d) Buffer level (e) Predicted buffer level (f) Resource constraints

Figure E.4: Typical simulation results for the 'ramp up' experiment

Experiment 3: Sine with extended storage capacity



(a) Number of exits (b) Cumulative backorders (c) Cumulative backorders MRP-C

(d) Buffer level (e) Predicted buffer level (f) Resource constraints

Figure E.5: Typical simulation results for the 'sine' experiment

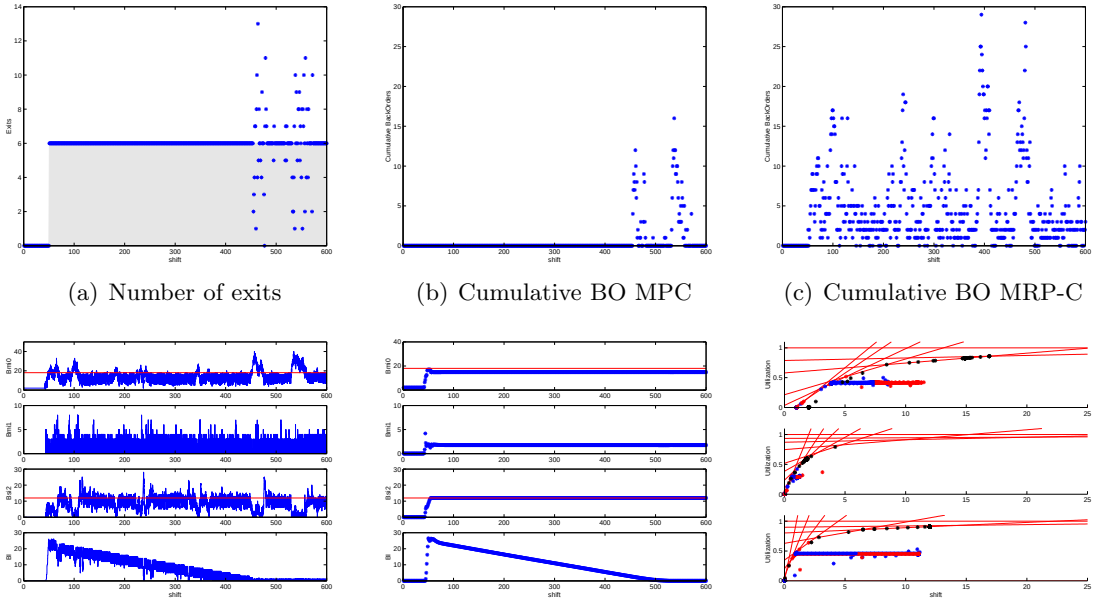
E.3 Simulation results Case II

In Section 5.4, the simulation results of the second case are discussed. The accompanying results are illustrated in this section. The simulation parameters of the conducted experiments are summarized in Table E.2.

	D_{ini}	D_{per}	D_{min}	D_{max}	sc_i	Run length	Planning horizon
1-a	50	0	0	6	(18, 12, 12)	600	630
1-b	50	0	0	6	(36, 24, 24)	600	630
2	50	150	0	6	(36, 24, 24)	600	630
3	48	96	0	6	(36, 24, 24)	600	630

Table E.2: Simulation parameters for the experiments of Case II

Experiment 1-a: Step up with original storage capacity

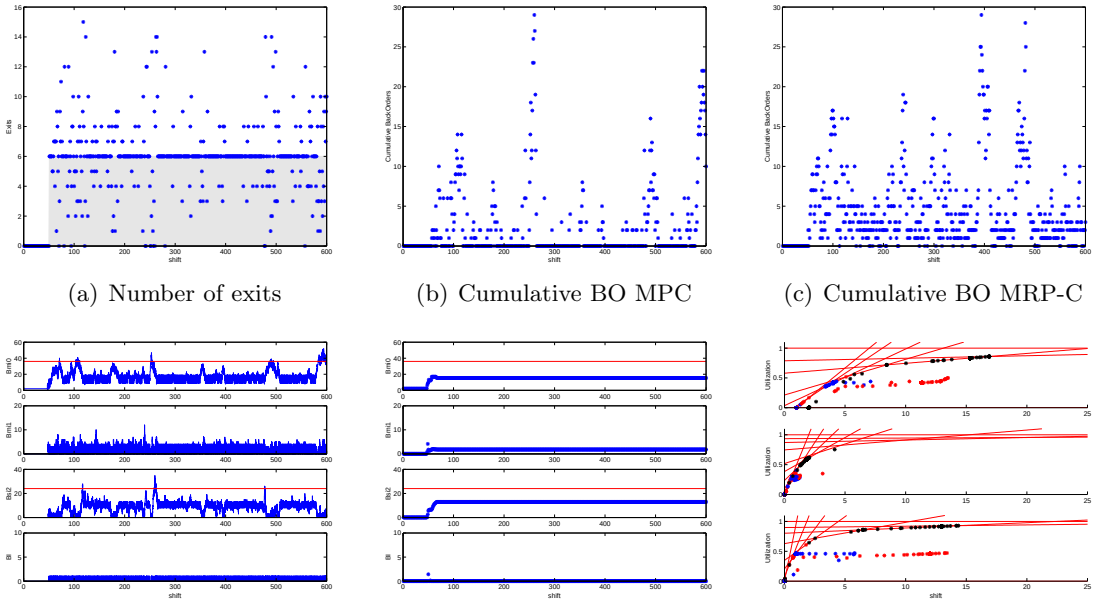


(a) Number of exits (b) Cumulative BO MPC (c) Cumulative BO MRP-C

(d) Buffer level (e) Predicted buffer level (f) Resource constraints

Figure E.6: Typical simulation results for the 'step up' experiment

Experiment 1-b: Step up with extended storage capacity

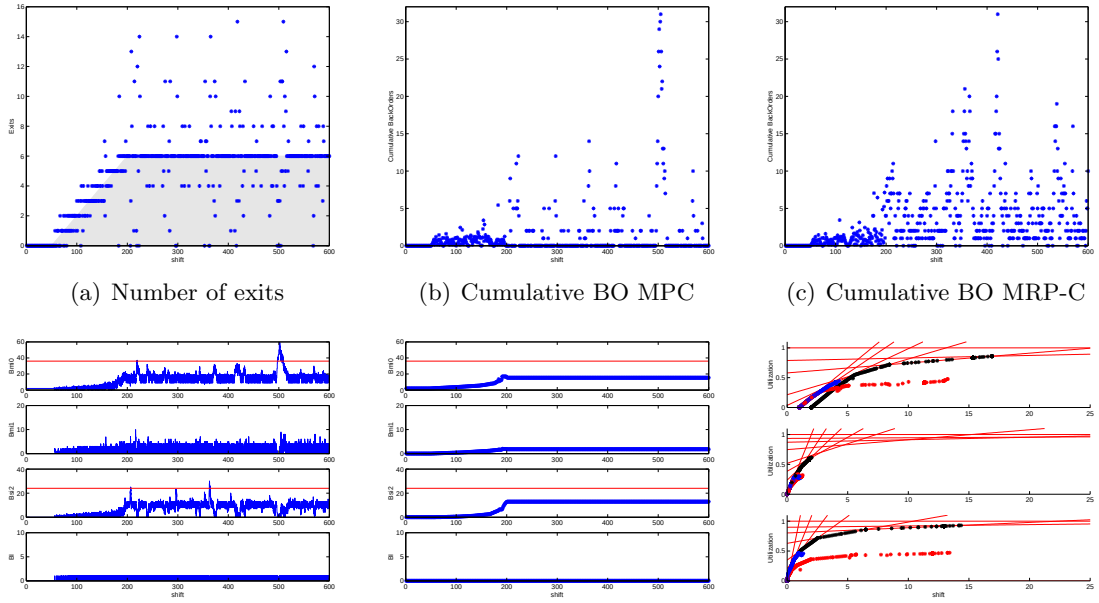


(a) Number of exits (b) Cumulative BO MPC (c) Cumulative BO MRP-C

(d) Buffer level (e) Predicted buffer level (f) Resource constraints

Figure E.7: Typical simulation results for the 'step up' experiment

Experiment 2: Ramp up with extended storage capacity

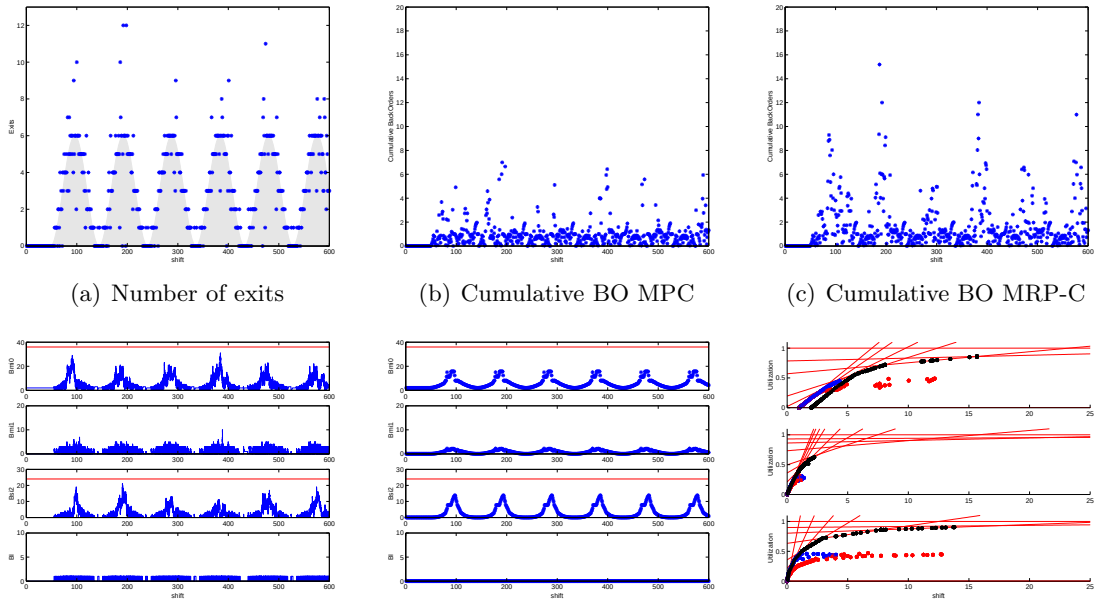


(a) Number of exits (b) Cumulative BO MPC (c) Cumulative BO MRP-C

(d) Buffer level (e) Predicted buffer level (f) Resource constraints

Figure E.8: Typical simulation results for the 'ramp up' experiment

Experiment 3: Sine with extended storage capacity



(a) Number of exits (b) Cumulative BO MPC (c) Cumulative BO MRP-C

(d) Buffer level (e) Predicted buffer level (f) Resource constraints

Figure E.9: Typical simulation results for the 'sine' experiment