TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Supervisory machine control by predictive-reactive scheduling

Citation for published version (APA):
Nieuwelaar, van den, N. J. M. (2004). *Supervisory machine control by predictive-reactive scheduling*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mechanical Engineering]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR581686

DOI:
10.6100/IR581686

Document status and date:
Published: 01/01/2004

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

Supervisory Machine Control by Predictive-Reactive Scheduling

Norbertus Josephus Martinus (Barend) van den Nieuwelaar

Voorkant: Het aardige van dit proefschrift is dat het voorgestelde besturingsconcept ook geïmplementeerd is, namelijk in de machine die op de voorkant is afgebeeld. T-ReCS is het acroniem waaronder het besturingsconcept bij ASML bekend is: Task-Resource Control System. Het DROSTE effect geeft aan dat T-ReCS op een gelaagde manier ingezet kan worden, waarbij taken vanuit een hogere laag (order)invoer zijn voor een lagere laag.

Cover: The nice thing about this thesis is the fact that the proposed control concept is actually implemented, namely in the machine depicted on the cover. T-ReCS is the acronym for the control concept as it is called at ASML: Task-Resource Control System. The "DROSTE" effect indicates that T-ReCS can be applied in a layered setting, where the tasks of a higher layer are (order) input for a lower layer.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).
IPA dissertation series 2004-21

© Copyright 2004, N.J.M. van den Nieuwelaar
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission from the copyright owner.

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Nieuwelaar, Norbertus J.M. van den

Supervisory machine control by predictive-reactive scheduling / by Norbertus
J. M. van den Nieuwelaar. - Eindhoven : Technische Universiteit Eindhoven, 2004.
Proefschrift. - ISBN 90-386-2756-4
NUR 804
Subject headings: supervisory machine control / predictive-reactive scheduling / exception recovery / kinematic calibration / deadlock avoidance / model checking / semiconductor equipment

Reproduction: Universiteitsdrukkerij Technische Universiteit Eindhoven

The work described in this thesis has been carried out at ASML in Veldhoven, the Netherlands. It contains Intellectual Property Rights of ASML. All rights reserved.

Supervisory Machine Control by Predictive-Reactive Scheduling

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van
de Rector Magnificus, prof.dr. R.A. van Santen,
voor een commissie aangewezen door het College
voor Promoties in het openbaar te verdedigen op
dinsdag 7 december 2004 om 16.00 uur

door

Norbertus Josephus Martinus van den Nieuwelaar
geboren te Tilburg

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. J.E. Rooda
en
prof.dr. J.C.M. Baeten

Copromotor:
dr.ir. J.M. van de Mortel-Fronczak

Preface

This thesis presents the results of four years of work in the field of supervisory machine control. It has been a challenge for me to serve two customers during this period: both the university and ASML. Looking back I get a satisfied feeling about what we achieved in the form of papers and this thesis on the one hand, and patent applications and applicability in practice on the other. I would like to use this preface to make some remarks about this thesis and to thank the people that contributed to my PhD project.

This thesis is a collection of papers. A consequence of choosing for this form is that there are some repetitions and that the last two papers have a different style and slightly different conventions. I did my best to relate the different papers in the introduction, and to conclude with an overall discussion of the applications and conclusions.

First of all, I would like to thank the management of ASML for giving me the opportunity to carry out this study while working for ASML. In particular I want to mention Tammo van den Berg, Harry Borggreve and Robbert van der Kruk. Furthermore, I want to thank my coaches at ASML, Rick van Lierop and Hans Onvlee, for helping me finding my way in ASML and for their useful comments.

Special thanks go to professor Koos Rooda. I hereby want to express my great gratitude for what we made happen together, and I am looking forward to continue this. Many thanks go to my co-promotor Asia van de Mortel for her coaching and especially for her help in writing things down. I also thank Cor Hurkens for his guidance in the area of scheduling. Furthermore I thank my second promotor, prof.dr. J.C.M. Baeten, and the other members of my reading committee, prof.dr. W.J. Fokkink and prof.dr. H. Nijmeijer, for their useful comments.

Besides the four years of work that I have put in this thesis, master students who helped me together have worked for a period more than double of that. Some of them are also co-authors of papers in this thesis: Roel Boumen, Niels Braspenning, Martin Driessen, Robert Dumont, and Michiel Stoets. Their main contribution was the implementation of the described functionality and working out the cases. Also Maarten van Bree, Suresh Punyamanthula, Stefan Roels, and Joris Vermunt did very valuable work. I am very grateful for the contribution of these master students to both the contents and the disclosure of this thesis, and to the great atmosphere in the T-ReCS team. I also thank the other co-authors of the papers in this thesis: Jan Friso Groote, Martijn Hendriks, and Frits Vaandrager, for enthusiastically picking up the challenges that I brought up and bringing in the computer science knowledge, mainly in the form of lemmas and proofs.

I also want to thank my colleagues both at the university and at ASML. From the university I want to mention Mieke Lousberg and thank her for her friendly help and

care. Moreover, I thank Albert Hofkamp and Ramon Schiffelers for their help and pleasant company. From ASML I thank all the people that contributed to this thesis, and especially the early adopters and propagators of T-ReCS: Ed de Gast, Peter van Gils, Koen van der Heijden, Wil Koenen, Raimond Visser, and Joost Worms. I want to thank Wil also for his help with the cover.

Finally, I like to thank my parents, other relatives, and friends. Last, but not least, I want to mention my wife with who I have two lovely kids: Ilse and Rick. Wendy, thank you for your love.

Summary

The subject of this thesis is supervisory control of complex manufacturing machines. ASML wafer scanners serve as carriers. A wafer scanner is a representative example of a complex manufacturing machine, containing many mechatronic systems. In complex manufacturing machines, many options exist to deploy the available resources to perform tasks that lead to the desired manufacturing purpose, resulting in various machine behaviors. Supervisory Machine Control (SMC) is responsible for deciding when to do which tasks using which resources.

The purpose of this project is to develop a suitable formal method for specification of supervisory control of complex manufacturing machines. There are some complicating requirements for SMC of complex manufacturing machines. First of all, the manufacturing tasks are heavily product recipe dependent, for which SMC must be flexible. This means that SMC must be able to handle a stream of mixed product types, which are being processed concurrently. Secondly, SMC must be able to optimize machine behavior by exploiting its resources in a best way possible within its manufacturing constraints. What is best, may depend on the characteristics of the recipe. Thirdly, SMC must fit in the dynamic environment that it is embedded in. This implies that it must react to all kinds of triggers from the environment without introducing unnecessary control overhead. A very important trigger is task failure: an exception, which requires a recovery reaction of SMC to avoid human intervention. Finally, in order to keep up with the increasing development pace in industry, SMC should allow easy adding and changing of functionality.

To fulfill the requirements mentioned above, a scheduling-based SMC concept is developed. To structure the control decisions to be made, a layered task resource framework is used. From an SMC point of view, a machine can be considered as a task resource system (TRS). Tasks can be associated with manufacturing processes, whereas resources can be associated with mechatronic systems. Transforming a manufacturing request into machine behavior can be structured in three phases, throughout which the constraints of the machine must be taken into account. First, a scheduling problem must be instantiated for the manufacturing request. This transformation is called *instantiating*. Subsequently, resources must be assigned to the tasks in the instantiated scheduling problem in some order, taking into account the fact that resources are able to perform certain tasks only, and only one at a time. This transformation is called *selecting*. The selected order of tasks to be performed by selected resources may imply consecutive state transitions of those resources. Finally, start and finish times can be assigned to the selected tasks, taking into account the durations of the tasks. This transformation is called *timing*. Combination of the selecting and timing transformation is referred to as scheduling. During the three

transformation phases of instantiating, selecting and timing, choices must be made. The consequences of a choice in a certain transformation on the machine behavior can only be evaluated by performing the consecutive transformations. Therefore, a transformation phase strongly relies on information from subsequent phases, which is expressed in the layered TRS framework. In this thesis, the three transformation phases are formally defined.

In this project, a predictive-reactive SMC framework has been developed that embeds the layered TRS framework in the form of TRS translation functions. Several methods or scenarios to react to different types of control triggers are described using these translation functions. For the 'nice weather' triggers it is important to avoid control overhead. This is done by making sure that reaction takes place in parallel with the manufacturing processes if possible. For control triggers involving exceptions it is more important to ensure robust recovery rather than to avoid control overhead. Therefore, reaction to exceptions is sequential. Basically, exception recovery uses the same transformation functions as the 'nice weather' triggers mentioned before, which is an elegant characteristic.

The scheduling transformation that is embedded in SMC uses heuristic filters to quickly find a good schedule. Several approaches are developed to avoid deadlocks. One of them uses a model checker to configure a least restrictive deadlock avoidance filter. A dedicated verification approach is developed to verify that the design of the filter configuration indeed cannot result in invalid machine behavior such as deadlock. This approach uses the specific structure of the scheduling model to apply state-space reduction techniques. These techniques make it possible to verify cases of practical size.

Two instantiating approaches are developed. The first one uses a database of instantiating rules and building blocks to generate a scheduling problem that fulfills the manufacturing request. The other approach uses meta-tasks and their pre-conditions and post-conditions to search for scheduling problem instances that fulfill the manufacturing request, like in game theory. This approach is also developed for kinematic calibrations. Kinematic calibration of high precision machines is in the scope of SMC as imperfections and drift effects of hardware must be corrected for during production.

Application of the proposed control concept has significant benefits compared to current practice at ASML. SMC related software development effort is expected to reduce by factor 2, and machine performance is expected to improve by order of magnitude 1 %. A limited version of the control concept has successfully been implemented in a part of the control software of the ASML TWINSCAN wafer scanner platform. A road map covering the evolutionary roll-out of the concept in the coming years is under development.

CONTENTS

CHAPTER

ONE

Introduction

This PhD study is practice driven: existing practical issues are to be addressed by any appropriate theory. As a consequence, the nature of the problems being faced in industrial practice determines the research direction and the applicable theory in the form of approaches, methods, and tools. Furthermore, focus is on the link between theory and practice rather than on theory itself. This implies emphasis on capturing practical phenomena in theory and enabling application of the theory in practice. This introduction starts with a description of the practical issues involved in complex manufacturing machines and the supervisory control associated with them. Wafer scanners [1] serve as carriers throughout this thesis. Inspired by the practical issues, the purpose of this study is formulated. After that, a framework that structures the research items is described, and the direction of the research is distilled. The framework is used to outline the theory that is developed and applied in this thesis addressing the research items.

1.1 Complex manufacturing machines

A wafer scanner is a representative example of a complex manufacturing machine. Wafer scanners are used in the semiconductor industry, and perform the most critical step in the manufacturing process of integrated circuits. Their primary manufacturing process is the exposure of an IC pattern onto a wafer, which is visualized in Fig. 1.1. Typically, the pattern is engraved on a so-called reticle. Light projects the pattern via a demagnification lens onto the wafer. During exposure the reticle and the wafer make a scanning movement, which explains the name wafer scanner. Exposure must be performed with very high accuracy. Therefore, reticles as well as wafers must undergo several pre-processing steps before exposure can take place. Pre-processing includes measuring of imperfections of the machine as well as the wafers and reticles to enable compensation for these imperfections.

To actually perform the processes, several mechatronic systems must be deployed. In Fig. 1.2 the main modules of the ASML TWINSCAN wafer scanner are pointed out. Typical for TWINSCAN is its dual wafer stage, enabling concurrent measurement and exposure of wafers. The modules consist of multiple mechatronic systems, together containing hundreds of sensors and actuators that can operate in parallel. A complex machine like a wafer scanner costs about 10^7 euro.

1.2 Supervisory control of manufacturing machines

In complex manufacturing machines, many options exist to deploy the available resources to perform tasks that lead to the desired manufacturing purpose, resulting in various machine behaviors. Supervisory Machine Control (SMC) is responsible for deciding when

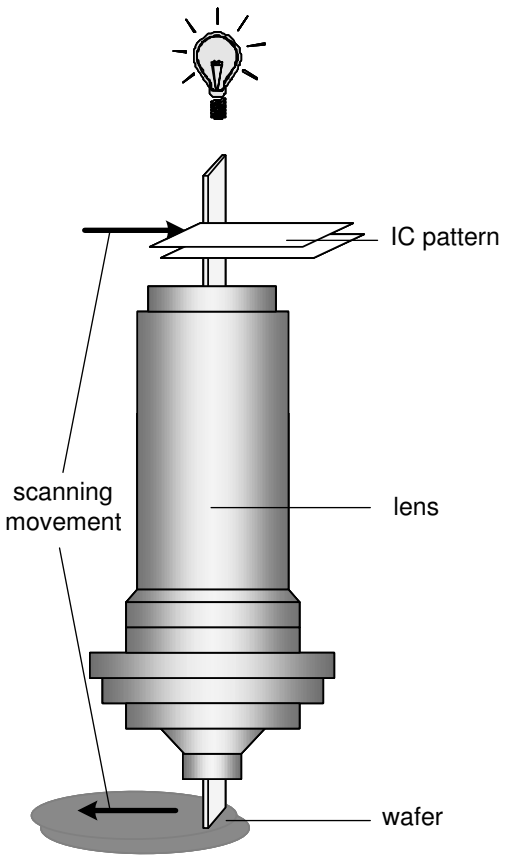


Figure 1.1: Exposure in a wafer scanner

to do which tasks using which resources such that the machine behaves as desired: task coordination. Regulative control in the resources is responsible for execution of the tasks received from SMC, and is not considered in this study. SMC is positioned in its context in Fig. 1.3.

The ASML wafer scanner control software is developed per product platform. An example of such a platform is the TWINSCAN platform. Platforms are developed in parallel and per platform several machine types and options exist that all are supported by the same embedded software package. A software package consists of order of magnitude 10^7 lines of C code, is deployed on approximately 10 processors and is developed and maintained by a few hundred software developers. About half of the code is application code and the rest is for support purposes like infrastructure. Although only one tenth of the application code of a software package can be associated with SMC, this part is one of the most complex parts whereas it largely determines the behavior of the machine.

Problems in current practice

The current practice in SMC of wafer scanners leaves considerable room for improvement. SMC can be characterized as rigid and increasingly complex. To start with, software complexity makes software development laborious and difficult to manage, leading to development efficiency loss. The most important consequence of this in the high-tech market of wafer scanners is the increase of time to market.

Furthermore, specification of SMC is often informal, which hinders analyzing the correctness of the design and the testing of the implementation. Even if a formal specification

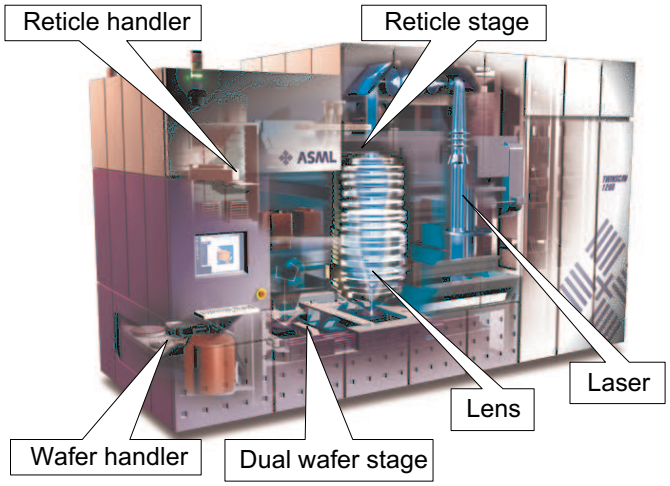


Figure 1.2: ASML TWINSCAN wafer scanner

is available, the transformation from specification to implementation in embedded software is done manually and therefore error-prone. Moreover, verification is hindered by state-space explosion. The effect of these shortcomings is revealed in an above-average number of problems per lines of code for SMC related software.

Finally, machine behavior cannot effectively be tailored to the specific demands of the product and customer. As a consequence, behavior is tuned for a typical setting. If machine behavior could be tailored better, the machines could better be fitted in the manufacturing process of more customers, which supports improving customer satisfaction and enlarging the customer base. Most importantly, tuning for a typical setting overall results in machine performance loss and a machine market value decrease.

Purpose of this project

The main purpose of the project described in this thesis is to develop a suitable formal framework for specification of supervisory machine control. Based on such a specification, analysis should be possible by means of simulation and verification. Moreover, such a specification should be the basis for the real supervisory machine control.

There are some important complicating requirements for SMC of complex manufacturing machines. First of all, manufacturing tasks as well as the definition of desired machine behavior heavily depend on product recipes, for which SMC must be flexible. Moreover, it must be able to handle a stream of mixed product types that are being processed concurrently. Secondly, it must fit in the dynamic environment that it is embedded in. This implies that it must react to triggers from the environment without introducing unnecessary control overhead. Especially reaction to tasks that fail (exceptions) is important. High-precision machines have a specific additional requirement. As mentioned before, they need to correct for geometric deviations originating from several sources during manufacturing. Finally, the trend towards ever shortening development cycles and increasing machine configuration variability demands that the machine behavior imposed by SMC is easily adaptable.

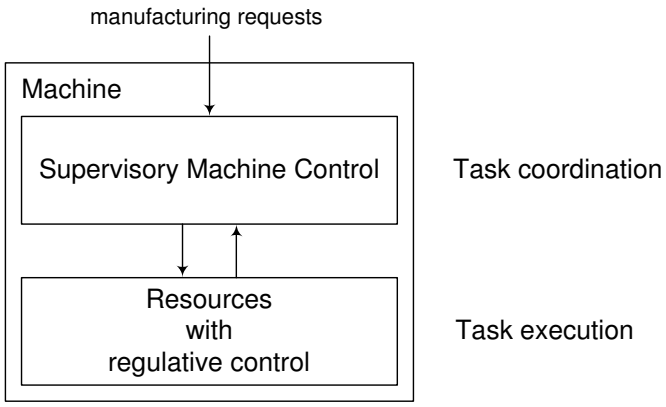


Figure 1.3: Context of Supervisory Machine Control

1.3 Task-resource view

This section briefly discusses supervisory control theory and scheduling theory to motivate the choice to base this thesis on the scheduling point of view. Subsequently, a layered Task Resource System (TRS) framework is introduced that forms the basis of this thesis. Furthermore, known techniques suitable in the context of this framework are discussed to motivate the research direction chosen in this thesis.

Supervisory control theory and scheduling theory

Wonham et al. [5, 11, 23] have developed a theory on supervisory control (SCT). In that theory, the system under control is described using Finite State Machines. The possible behavior of such a system is regarded as a language. A supervisory controller in the form of a deterministic automaton is synthesized that restricts the language by disabling a subset of events, to control the system to properly accomplish its task. This implies that supervisors must be modelled specifically for the task to be accomplished. Unfortunately, this does not meet the first requirement mentioned earlier. SMC for complex manufacturing machines must be more flexible as the manufacturing tasks are heavily dependent on the particular recipe and, therefore, differ per manufacturing request. Moreover, the requirements of an optimal controller differ per recipe.

From the scheduling point of view, the tasks to be performed under the restrictions imposed by the machine resources can be regarded as a scheduling problem, as in [7, 15, 24, 25]. Per manufacturing request the task-related part of the scheduling problem must be instantiated by SMC. The recipe-dependent definition of desired behavior translates to run-time scheduling with recipe-dependent optimization criteria [19]. The fact that SCT lacks the required flexibility that nicely matches the scheduling-based approach made us decide to base this project on the scheduling point of view in which tasks and resources play a prominent role.

Layered Task Resource System framework

From the SMC point of view, a machine can be considered as a TRS. Transforming a manufacturing request into machine behavior can be structured in three phases. First, a scheduling problem must be instantiated (defined) from the manufacturing request, taking

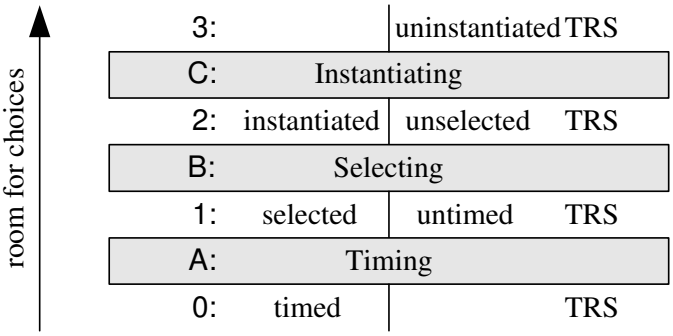


Figure 1.4: Layered Task Resource System framework

into account the limitations of the machine. This transformation is called *instantiating*. The structure of the resulting scheduling problem shows many similarities with the job shop scheduling problem [22]. The manufacturing process of a material instance can be associated with a job, whereas the different parallel mechatronic systems can be associated with the different machines in a job shop. Subsequently, resources must be assigned to the tasks in the instantiated scheduling problem in some order, taking into account the fact that resources are able to perform certain tasks only, and only one at a time. This transformation is called *selecting*. The selected order of tasks to be performed by selected resources implies consecutive physical state transitions of those resources, which is analogous to the setup times for mode switching in job shop scheduling. Finally, start and finish times can be assigned to the tasks, taking into account the physical restrictions of the resources. This transformation is called *timing*.

During the three transformation phases of instantiating, selecting and timing, choices must be made. The result of a choice in a certain transformation on the machine behavior can only be evaluated by performing the consecutive transformations. Therefore, a transformation phase strongly relies on subsequent phases, and this is analogous to the *Layers* architectural pattern described in [6]. The layered TRS framework shown in Fig. 1.4 displays the hierarchically related transformation phases as functionality layers (A through C) and the TRS definitions of different levels (0 through 3) as interfaces between the layers. This framework was first presented in [21].

Discussion of techniques

This subsection discusses known techniques suitable to perform analysis based on the system definitions characterized in the previous subsection, and is based on [21].

A task resource system can be classified as a hybrid system in the sense that it contains both continuous-time and discrete-event characteristics. In computer science, several generic hybrid paradigms and associated languages exist which are accompanied by various analysis tools [4, 12]. After a mapping of the original system onto such a paradigm and language, timing analysis can be performed. Supporting tools can be classified as either model checkers or simulators, in case of exploration of the complete state space (all realizations) or exploration of just any realization, respectively. An essential part of the derivation of such a realization is finding time-optimal trajectories, which is very complicated in general. Therefore, a simulator is not suitable for derivation of

the minimal duration. A model checker is able to determine whether a certain property holds. This property could be whether a solution exists that takes no more than a certain amount of time. Embedding a model checker in an optimization algorithm that iterates towards the optimal solution would be a possible though inefficient solution to finding a time-optimal trajectory. Some model checkers have limited optimization extensions.

However, from a supervisory machine control point of view, only resource start and end states are considered per task. Only the duration of a task matters, not the state trajectory. Therefore, abstraction from continuous behavior is possible. Assuming that the duration of the state transitions is known, the model can be simplified to the class of discrete-event systems. In wafer scanners, determination of the required duration of the resource state transition can be addressed in a pragmatic way. For the majority of the resources, SMC requires only a finite set of state transitions. For these state transitions, a table of state transition durations can be determined in several ways. Some resources have to support an infinite number of state transitions: the stages. Determination of their duration can be performed by combining analytical functions for most cases. Only in very special cases, an approximation algorithm is required. As this concerns a very restricted solution area, a very simple bisection algorithm suffices [16]. These dedicated mathematical functions take less computing power to find a solution than a generic solver.

Also for discrete-event systems, a wide range of paradigms and languages exists, e.g. [3]. Supporting tools can be classified analogous to the tools supporting hybrid languages. Because there are no alternatives in case of a completely predefined system, both tool classes are suited for analysis. This leaves the disadvantage of mapping the original system onto the generic discrete-event paradigm and language. Even this mapping can be prevented, as calculation of the minimal time to execute a selected untimed TRS (level 1) with given resource state transition durations is in fact a linear programming (LP) problem [28]. This enables the usage of a variety of mathematical tools, to derive the total system behavior (level 0) from the durations of resource state transitions. In case of dynamic scheduling, it is desirable that computation of system behavior starts with tasks that can be dispatched first, called forward computation. In this way, some tasks can already be released, while timing determination of the rest of the tasks is still under progress. Forward computation is not applicable for generic LP problem solvers. A mathematical approach for which this is applicable is the Heaps of Pieces approach [27]. A restriction of this approach is that it cannot cope with precedence relations, that are common in an unselected TRS (level 2).

TRS definitions of level (2) and (3) can have several realizations. Therefore, in principle a simulator is not suited for analysis, whereas a model checker is because it evaluates all possible realizations. However, the combination of possible choices blows up the number of realizations exponentially. In model-checking terminology, this phenomenon is known as state-space explosion. Furthermore, the mapping of the original model onto a generic paradigm and language usually introduces even more possibilities than exhibited by the original model. Therefore, model checkers are in principle not suited to analyse practical cases. In operations research, several approaches can be found that address certain aspects of the choices from alternatives. The choice of tasks and task precedences for one resource results in different realization times of a certain scheduling problem instance. This issue is widely discussed in literature, and is referred to as the Travelling Salesman Problem [17], or more specific: the Rural Postman Problem or the Vehicle Routing Problem [26]. Because only one resource is considered in those approaches, there is no parallelism. The choice from resource alternatives and task precedence alternatives for

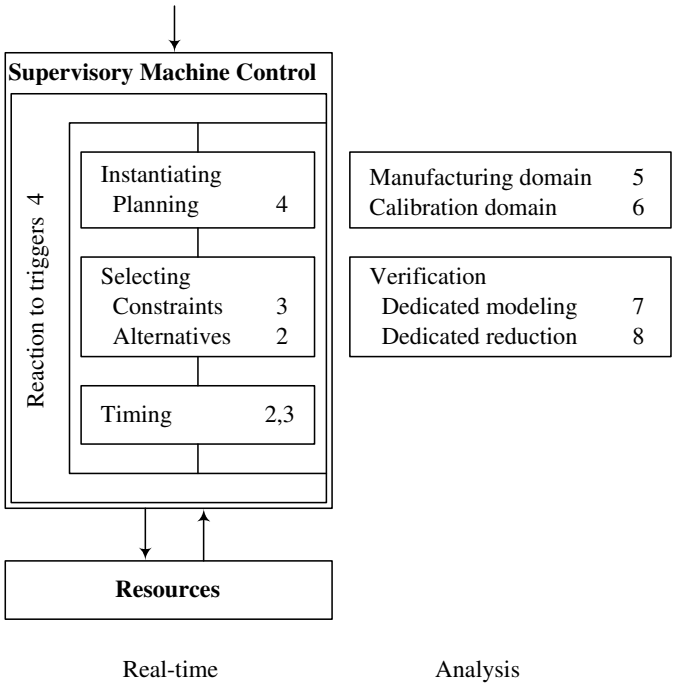


Figure 1.5: Projection of the main contributions of the chapters of this thesis onto the TRS framework

resources is also widely discussed in literature, and is referred to as the (Generalized) Job Shop Scheduling problem (JSS) [28]. The fact that in a complex manufacturing machine multiple tasks may exist that have the same effect gets little attention in scheduling literature. The same goes for machine-specific scheduling constraints imposed by the tight physical space in a machine.

The precedence relation in a TRS of level 2 assumes that the related tasks are performed successfully. When considering exceptional behavior, construction of a recovery must be based on the constraints of the machine only: TRS level 3. These constraints are analogous to rules of games [2]. Application of game theory [10] in this domain is unknown.

Usage of predictions based on a model in a control system is known in control theory literature as model-based predictive control [18]. The underlying line of thoughts can also be used in supervisory control. In some cases, a well-founded decision can only be made if future behavior can be predicted, for instance, by a model of the system that is embedded in supervisory control.

In this thesis, the control decision-making process of SMC is specified by making the TRS definitions and transformations shown in Fig. 1.4 explicit. Besides that, the TRS framework is embedded in the real-time environment of SMC such that efficient and effective reaction to triggers results.

1.4 Outline and context

This thesis is a collection of articles. In the sequel, the main contributions of the chapters are projected onto the TRS framework and their coherence is described, which is summarized visually in Figure 1.5.

Backbone

The backbone of the thesis is in Chapters 2 through 4, describing a scheduling-based SMC concept.

Chapters 2 and 3 discuss the timing and the selecting functionality and complete the definition of the static scheduling problem of transforming a TRS definition of level 2 into a TRS definition of level 0: timed machine behavior. Existing scheduling theory is used where possible, and extensions have been introduced where necessary. Extensions were necessary in two areas. One area is the definition of the room for choice of tasks, which is described in Chapter 2. The other area is the definition of the machine-specific constraints implied by the physical restrictions in a machine. These restrictions have to do with material logistics and resource interference, and are described in Chapter 3. The material logistic restrictions can cause the machine to deadlock. Additional constraints are described to avoid deadlock. Besides the definition of the scheduling problem, a predictive scheduling approach is proposed and a scheduling algorithm that is suited for application in SMC is described.

Chapter 4 embeds the developed scheduling functionality in the dynamic environment of SMC, encompassing instantiating and reaction to triggers from the environment. A straightforward instantiating functionality, called planning, is described: rule-based construction of a TRS definition of level 2 from predefined building blocks. The SMC concept described consists of a predicting part that embeds the TRS framework and a real-time dispatching part that is connected to the resources. Several scenarios to react to different types of triggers (including exceptions) are described that enable efficient and effective response. This completes the concept of SMC by predictive-reactive scheduling.

Instantiating constraints

Chapter 5 describes the combinatoric effect that makes it practically impossible to predefine all recovery scenarios. This is a disadvantage of instantiating a TRS definition using planning. Planning is based on predefined building blocks and construction rules instead of the TRS definition of level 3 in Figure 1.4. A solution to this is to search run-time for an exception recovery within the essential constraints imposed by the machine resources, which is explored in Chapters 5 and 6. Chapter 5 captures the instantiating constraints for general manufacturing processes and material transport in a TRS definition of level 3. Moreover, it proposes a search algorithm that can be applied to search for exception recoveries.

Chapter 6 specifically focusses on the domain of calibration sequences in high-precision machines. To capture the instantiating constraints, analysis of the geometric relations and inaccuracies in the machine is necessary. This can be associated with theory on kinematic calibration [14]. The main contribution of Chapters 5 and 6 lies in the explicit capturing of the phenomena that constrain instantiation.

The primary purpose of these chapters is analysis. Embedding of the instantiating functionality in SMC receives little attention.

Verification

Although the computer science part of the project described in [20] has not fully been addressed, some aspects concerned with verification have been investigated in co-operation

with the computer science departments of Radboud University Nijmegen and Eindhoven University of Technology.

The predicting part of the described SMC concept can be used for analysis by simulation. Chapters 7 and 8 describe the mechanical engineering point of view concerning analysis by verification. The machine-specific constraints captured in the scheduling problem (TRS definition of level 2) ensure feasible behavior of the machine. In addition, an essential property to guarantee is absence of deadlocks. Furthermore, it is desirable to guarantee that the machine behaves in a time-optimal way. Such properties can be verified using model checking [8], which is hindered by state-space explosion in industrial-sized cases like this. Chapters 7 and 8 present two approaches that use specific information of the property to be verified and the structure of the model to cope with this problem.

Chapter 7 focusses on a specific class of systems that can be described by a TRS definition of level 2. The most important restriction is that all considered products are processed in the same way: no recipe dependency. For this class of systems a least restrictive deadlock avoidance policy is synthesized and checked. As opposed to the deadlock avoidance constraints described in Chapter 3, this policy does not exclude more schedules than necessary by also taking the transport direction of the material instances into account. The approach results in an expression characterizing the safe states of the machine. This expression can be applied in the scheduling approach described in Chapter 3, analogous to the other deadlock avoidance constraints. Deadlock is avoided by keeping the machine in a safe state. Furthermore, time optimality of steady-state operation is checked in Chapter 7. Two consistent models of different abstractions are used to verify the two properties.

Chapter 8 considers the entire class of systems that can be defined by a TRS definition of level 2, but is limited to the verification of absence of deadlocks. The specific structure of the model is used to reduce the state space such that verification of the specific property is still possible.

These approaches are in accordance with the message of the *No Free Lunch Theorem* [29]: in combinatoric problems, dedicated approaches perform better than generic approaches. For formal proofs related to verification we refer to the original computing science documents [9, 13].

After the chapters describing the theory that has been applied and developed, applications of the theory in practice are described in Chapter 9. Chapter 10 concludes this thesis with a discussion wrapping up the contributions of this project and suggestions for further research.

References

[1] ASML, 2004. Information on wafer scanners available through URL `http://www.asml.com/`, item: products - lithography.

[2] R. J. Aumann and S. Hart. *Handbook of game theory: with economic applications*. Amsterdam, North-Holland, 2002.

[3] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Number 18. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1990.

[4] D. A. van Beek and J. E. Rooda. Languages and applications in hybrid modelling and simulation: positioning of Chi. *Control Engineering Practice*, 8(1):81–91, 2000.

[5] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–341, 1994.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, 1996.

[7] H. Chen and Baosheng Hu. Schedule-driven supervisory control of flexible manufacturing systems. In *30th Conference on Decision and Control*, pages 2186–2191, 1991.

[8] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.

[9] M. M. H. Driessen. Verification of task resource scheduling, June 2004. Internship report of Department of Computer Science, Eindhoven University of Technology, The Netherlands, available through URL `http://se.wtb.tue.nl/~bvdnieuw`.

[10] A. Garnaev. *Search games and other applications of game theory*. Springer, 2000.

[11] P. Gohari and W. M. Wonham. Reduced supervisors for timed discrete-event systems. *IEEE Transactions on Automatic Control*, 48(7):1187–1198, 2003.

[12] H. Gueguen and M. Lefebvre. A comparison of mixed specification formalisms. In *Automation of mixed processes: Hybrid Dynamic Systems: ADPM 2000*, pages 133–138, Aachen, 2000. Shaker Verlag.

[13] M. Hendriks, N. J. M. van den Nieuwelaar, and F. W. Vaandrager. Model checker aided design of a controller for a wafer scanner. Report NIII-R0430, Nijmegen Institute for Computing and Information Sciences, University of Nijmegen, The Netherlands, June 2004.

[14] J. M. Hollerbach. *A survey of kinematic calibration*. MIT Press, 1989.

[15] J. Kim, T. Lee, H. Lee, and D. Park. Scheduling analysis of time-constrained dual-armed cluster tools. *IEEE Transactions on Semiconductor Manufacturing*, 16(3):521–534, 2002.

[16] C. M. H. Kuijpers, C. A. J. Hurkens, and J. B. M. Melissen. Fast movement strategies for a step-and-scan wafer stepper. *Statistica Neerlandica*, 51(1):55–71, 1997.

[17] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley-Interscience, Chichester, 1985.

[18] J. M. Maciejowski. *Predictive control with constraints*. Prentice Hall, Harlow, 2002.

[19] H. Marchand, O. Boivineau, and S. Lafortune. On the synthesis of optimal schedulers in discrete-event control problems with multiple goals. *SIAM Journal on Control Optimization*, 39(2):512–532, 2000.

[20] N. J. M. van den Nieuwelaar. Project plan: A framework for development of machine control systems, November 2000. Available through URL `http://se.wtb.tue.nl/~bvdnieuw`.

[21] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, and J. E. Rooda. Design of supervisory machine control. In K. Glover and J. Maciejowski, editors, *Proceedings of the European Control Conference 2003*, 2003. CD-ROM.

[22] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 1995.

[23] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[24] S. Rostami and B. Hamidzadeh. Optimal scheduling techniques for cluster tools with process-module and transport-module residency contraints. *IEEE Transactions on Semiconductor Manufacturing*, 15(3):341–349, 2002.

[25] Y. Shin, T. Lee, J. Kim, and H. Lee. Modeling and implementing a real-time scheduler for dual-armed cluster tools. *Computers in Industry*, (45):13–27, 2001.

[26] P. Toth and D. Vigo. *Predictive control with constraints*. SIAM, Philadelphia, 2002.

[27] G. X. Viennot. Heaps of Pieces, I: Basic definitions and combinatorial lemmas. In G. Labelle and P. Leroux, editors, *Combinatoire Enumerative*, pages 321–350. Springer, New York, 1986.

[28] M. Wennink. *Algorithmic Support for Automated Planning Boards*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1995.

[29] D. H. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

CHAPTER

TWO

SCHEDULING ALTERNATIVES AND ALGORITHM

This chapter contains the paper *Predictive Scheduling in Complex Manufacturing Machines: Scheduling Alternatives and Algorithm* that has been protected in patent application ASML ref. P-1784. First filing was in the US at December 23, 2003, number 10/743,320.
The paper was submitted to IEEE Transactions on Automatic Control in February 2004.

Predictive scheduling in complex manufacturing machines: scheduling alternatives and algorithm

N.J.M. van den Nieuwelaar †*, J.M. van de Mortel-Fronczak †,
N.C.W.M. Braspenning †, J.E. Rooda †

Abstract

Supervisory control of a complex manufacturing machine - which involves coordination of many mechatronic systems - requires proper scheduling. Supervisory control must be flexible to concurrently process a mix of different product types each requiring heavily recipe-dependent manufacturing tasks, without introducing unnecessary control overhead. Analysis of the alternatives to choose from is done using an example case, a wafer scanner. The job shop scheduling model is suited to define alternatives concerning task order and resource assignment. This definition is extended for alternatives with respect to tasks, as different tasks can lead to the required manufacturing result. To formally describe the scheduling process that transforms the model into a certain behavior, the transformation is split into two phases: selecting and timing, according to a layered task resource system framework. Constraints are formulated for each transformation phase and a scheduling algorithm is described that is suited for usage in supervisory machine control. In the algorithm, the two transformations are closely interweaved into a constructive algorithm allowing early dispatching of the schedule. For efficiency, a compact data structure is used to represent the choices made: heaps of pieces. Results show intuitive modelling of the scheduling alternatives and effective machine behavior optimization.

2.1 Introduction

The purpose of a manufacturing machine is to make products, which requires physical manufacturing processes to be carried out. To actually do the work, mechatronic systems in the machine must be deployed. Control in the separate mechatronic systems is referred to as low-level control and is not considered in this chapter. In complex manufacturing machines, many options exist to deploy the available resources to perform tasks that lead to the desired manufacturing purpose, resulting in various machine behaviors. Supervisory Machine Control (SMC) is responsible for deciding when to do which tasks using which resources. There are three important complicating requirements for SMC of complex manufacturing machines. First of all, the manufacturing tasks are heavily product recipe dependent, for which SMC must be flexible. Furthermore, it must be able to handle a stream of mixed product types, which are being processed concurrently. Finally, no unnecessary control overhead may be introduced.

† Eindhoven University of Technology: P.O. box 513, 5600 MB Eindhoven, The Netherlands.
* ASML: De Run 6501, 5504 DR Veldhoven, The Netherlands.
Corresponding author: N.J.M. van den Nieuwelaar, e-mail: n.j.m.v.d.nieuwelaar@tue.nl

2.1.1 Literature

Many approaches exist to describe a system under supervisory control using well-known formalisms from computer science. Supervisory control theory as discussed by Wonham et al. [2, 5, 13] models the system under control using Finite State Machines. The possible behavior of such a system is regarded as a language. A supervisory controller in the form of a deterministic automaton is synthesized that restricts the language by disabling a subset of events, to control the system to properly accomplish its task. Supervisors must be modelled specifically for the task to be accomplished, and therefore are not flexible for handling different recipes.

Literature on performance analysis and supervisory control of complex manufacturing machines that can handle different recipes [7, 14, 15] encapsulates models of the tasks to be done for different recipes, which can be associated with scheduling theory. However, in this work only part of the total problem is analyzed. Some of the work is restricted to steady state behavior. As in complex manufacturing machines the time spent for transient behavior (e.g. while switching product types) is of the same order of magnitude as steady-state behavior, it is of importance not to focus on steady-state behavior only. Other work is restricted with respect to other areas, for example a single resource [8] or a subset of tasks [6], whereas SMC is responsible for control of the entire machine. Moreover, the dynamic and real-time circumstances that SMC must operate in should get sufficient attention, which is often underexposed in scheduling literature [16].

To properly address the recipe-dependent tasks to be done, an approach based on a well-known scheduling problem is proposed in this chapter, as is also done in [10].

2.1.2 Layered task resource system framework

From a SMC point of view, a machine can be considered as a task resource system (TRS). Tasks can be associated with manufacturing processes, whereas resources can be associated with mechatronic systems. Transforming a manufacturing request into machine behavior can be structured in three phases. First, a scheduling problem must be instantiated from the manufacturing request, taking into account the limitations of the machine. This transformation is called *instantiating*. The structure of the resulting scheduling problem shows many similarities with the job shop scheduling problem [12]. The manufacturing process of a material instance can be associated with a job, whereas the different parallel mechatronic systems can be associated with the different machines in a job shop. Subsequently, resources must be assigned to the tasks in the instantiated scheduling problem in some order, taking into account the fact that resources are able to perform certain tasks only, and only one at a time. This transformation is called *selecting*. The selected order of tasks to be performed by selected resources implies consecutive state transitions of those resources, which is analogous to the setup times for mode switching in job shop scheduling. Finally, start and finish times can be assigned to the tasks, taking into account the speed of the resources. This transformation is called *timing*.

During the three transformation phases of instantiating, selecting and timing, choices must be made. The result of a choice in a certain transformation on the machine behavior can only be evaluated by performing the consecutive transformations. Therefore, a transformation phase strongly relies on information from subsequent phases. The layered TRS framework shown in Fig. 2.1 displays the hierarchically related transformation phases as functionality layers (A through C) and the different TRS definition levels (0 through 3) as interfaces between the layers (see Chapter 1 and [11]).

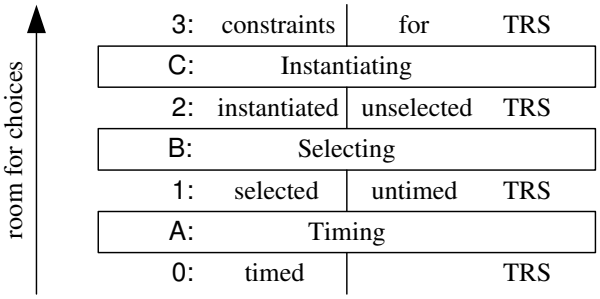


Figure 2.1: Layered Task Resource System framework

2.1.3 Predictive scheduling in Supervisory Machine Control

This chapter discusses predictive scheduling [17] in SMC based on task-resource system definitions. Scheduling can be associated with transforming a TRS definition of level 2 into a TRS definition of level 0, involving the layers A and B shown in Fig. 2.1. Several motivations exist for predictive scheduling in SMC of complex machines. First, making choices run-time enables optimized machine behavior for different products and users. Several optimization approaches can be applied: heuristics can be used for guidance and several schedules can be generated in several ways to do 'what-if' analysis. Second, the TRS definitions used are suited for design-time analysis and any error-prone gap between the design and the implementation is minimized as run-time execution is based on the same model. Finally, predictive scheduling is flexible, which eases adaptation to the evolution of machine configurations or machine operation philosophies. A potential drawback of predictive scheduling is control overhead, which slows down the machine. To minimize this effect, a constructive scheduling algorithm can be applied that allows for partial schedule dispatching.

2.1.4 Structure of the chapter

This chapter is structured as follows. Throughout the chapter, an example of a complex machine is used for illustration: a dual-stage wafer scanner [1]. Other examples of complex manufacturing machines in the semiconductor industry are cluster tools and tracks. Section 2.2 describes the wafer scanner example and how some of the scheduling alternatives can be defined in the form of a generalized job shop scheduling problem. It is pointed out that in complex manufacturing machines also choices with respect to tasks play a role, which do not map onto the job shop scheduling problem. Analysis is done using the example to determine the required expressivity for this type of scheduling alternatives. Section 2.3 describes the additional TRS definition level 2 elements and the concerned selecting constraints that can express the room for choices with respect to tasks. Using these additional elements and constraints, the total scheduling problem is defined. In Section 2.4, the transformations selecting and timing are integrated into a scheduling algorithm that is suited for usage in a run-time environment. Section 2.5 shows an example manufacturing scenario for the wafer scanner to which the scheduling algorithm is applied. Using two different heuristic settings, its potential for machine behavior optimization is shown. Finally, concluding remarks are presented in Section 2.6.

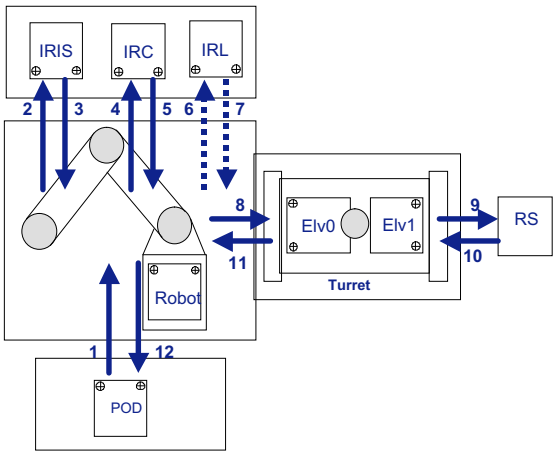


Figure 2.2: Reticle handling tasks

2.2 Scheduling in a dual-stage wafer scanner

2.2.1 A dual-stage wafer scanner

In this subsection, a dual-stage wafer scanner is described. The primary manufacturing process of a wafer scanner is the exposure of a mask containing an IC pattern onto wafers. In a dual-stage wafer scanner, two wafer stages are available to carry wafers during exposure. Typically, a few hundred integrated circuits or dies are placed on a wafer of 300 mm diameter, and for the exposure step each of them must be scanned in either direction: up or down. A mask is placed on a reticle, which is contained in a POD (see Fig. 2.2) and needs to be pre-processed prior to being used for exposure at the reticle stage (RS). Important pre-processing steps for reticles are cleaning (at IRC) and inspecting (at IRIS), and for wafers aligning and measuring. Measuring encompasses scanning a minimum number of mark pairs which are distributed over the wafer. In between the processing steps, several logistic steps must take place, for instance, via the Turret that consists of two elevators (Elv0 and Elv1). These logistic steps are illustrated for reticles in Fig. 2.2 using numbered arrows. The dotted arrows concern the optional usage of a buffer (IRL, tasks 6 and 7). Each resource is denoted by a square. Several wafers and reticles are concurrently being processed, and their logistic paths can cross each other like at the robot of Fig. 2.2. For instance at the robot, sequence dependent state transitions (rotations) must be performed when switching from one reticle to another. The number and layout of the dies and mark pairs on the wafer, as well as process speeds are determined by the product recipes.

2.2.2 Job Shop Scheduling

In this subsection, job shop scheduling starting from an instantiated, unselected TRS definition (level 2 in Fig. 2.1) is formally defined. First, the instantiated, unselected TRS definition of a job shop scheduling problem is described, which subsequently is extended following generalized job shop scheduling. After that, the selected, untimed TRS definition is described, and the constraints to be taken into account when selecting from the alternatives defined in definition level 2 to reach definition level 1. Finally, the timed TRS definition (level 1) and the constraints concerning transformation A are summarized.

A job shop scheduling model can be defined by a 6-tuple in terms of tasks and resources $(\mathcal{T}_2, \mathcal{R}, I_2, P_2, Sb_2, Se_2)$ in which:

- $\mathcal{T}_2$ is a finite set of elements called tasks.
- $\mathcal{R}$ is a finite set of elements called resources.
- I_2: $\mathcal{T}_2 \rightarrow \mathcal{R}$ gives the resource that is involved in a certain task.
- $P_2 \subseteq \mathcal{T}_2 \times \mathcal{T}_2$ is the precedence relation between tasks.
- $Sb_2, Se_2 : TR \rightarrow \mathcal{S}$ give the begin and the end (physical) state of the resource involved in a certain task, where $TR = \{(t, r) | t \in \mathcal{T}_2, r = I_2(t)\}$.

Note that, by convention, the definition level is added to each element in subscript. Elements that are not level-specific have no subscript.

In the sequel, several constraints are defined and labelled according to the following convention. Each constraint label starts with a C- followed by the transformation layer identifier (A, B, C) or definition level identifier (0, 1, 2, 3) and subsequently by a letter (a, b, ...).

The constraint that must be satisfied for the instances of the definition elements is as follows:

C-2a P_2 contains no cycles.

However, like in the wafer scanner example, in a complex machine multiple resources can exist which all are capable of the same work. Furthermore, some tasks involve synchronous transitions of multiple resources, e.g. a scan. As this can also be the case in job shops, the job shop scheduling problem has been generalized to incorporate these features [19].

A generalized job shop scheduling problem can be defined by an 8–tuple $(\mathcal{T}_2, \mathcal{R}, \mathcal{C}, I_2, A, P_2, Sb_2, Se_2)$:

- $\mathcal{T}_2$ is a finite set of elements called tasks.
- $\mathcal{R}$ is a finite set of elements called resources.
- $\mathcal{C}$ is a finite set of elements called capabilities.
- I_2: $\mathcal{T}_2 \rightarrow \mathcal{P}(\mathcal{C})$ gives the set of capabilities that are involved in a certain task.
- A: $\mathcal{C} \rightarrow \mathcal{P}(\mathcal{R})$ gives the set of resources that are available for a certain capability.
- $P_2 \subseteq \mathcal{T}_2 \times \mathcal{T}_2$ is the precedence relation between tasks.
- $Sb_2, Se_2 : TC \rightarrow \mathcal{S}$ give the begin and the end (physical) state of each capability involved in a certain task, where $TC = \{(t, c) | t \in \mathcal{T}_2, c \in I_2(t)\}$.

Constraint C-2a remains.

The selecting transformation (B) assigns resources to tasks, and determines the order in which tasks are executed for each resource. By this transformation, an unselected TRS is transformed into a selected, untimed TRS, which can be defined by a 6-tuple $(\mathcal{T}_1, \mathcal{R}, I_1, P_1, Sb_1, Se_1)$:

- $\mathcal{T}_1$ is a finite set of elements called tasks.
- $\mathcal{R}$ is a finite set of elements called resources.
- I_1: $\mathcal{T}_1 \rightarrow \mathcal{P}(\mathcal{R})$ gives the set of resources that are involved in a certain task.
- $P_1 \subseteq \mathcal{T}_1 \times \mathcal{T}_1$ is the precedence relation between tasks.
- $Sb_1, Se_1 : TR \rightarrow \mathcal{S}$ give the begin and the end (physical) state of the resource involved in a certain task, where $TR = \{(t, r) | t \in \mathcal{T}_1, r \in I_1(t)\}$.

Constraints that have to be satisfied for the instances of the definition elements are as follows:

C-1a The sequence of tasks per resource is a chain.

Constraints that have to be satisfied for the selecting transformation can be formulated as follows:

C-Ba The sequence of selected tasks per resource is a chain (similar to C-1a).

C-Bb For each selected task, an available resource must be selected for each involved capability:
$(\forall t, r, c : t \in \mathcal{T}_1, r \in I_1(t), c \in I_2(t) : r \in A(c))$

The begin and end state definition is obtained from the capability of the selected resource.

By the timing transformation (A), a selected, untimed TRS is transformed into a timed TRS, which can be defined by a 5-tuple $(\mathcal{T}_0, \mathcal{R}, I_0, \tau_{S_0}, \tau_{F_0})$:

- $\mathcal{T}_0$ is a finite set of elements called tasks.
- $\mathcal{R}$ is a finite set of elements called resources.
- I_0: $\mathcal{T}_0 \rightarrow \mathcal{P}(\mathcal{R})$ is the set of resources that are involved in a certain task.
- τ_{S_0}, τ_{F_0}: $\mathcal{T}_0 \rightarrow \mathbb{R}^+$ are the start time and the finish time of a certain task, which implies that all resources assigned to a task t are occupied for the same time span.

Furthermore, note that a timed TRS can be visualized as a Gantt chart.

The constraint that has to be satisfied for the instances of the definition elements is as follows:

C-0a Per resource there is a chronological sequence of pairs of task start and task finish times.

Constraints that have to be satisfied for the timing transformation are as follows:

C-Aa Nothing changes with respect to tasks and the (involved) resources: $\mathcal{T}_0 = \mathcal{T}_1, I_0 = I_1$

C-Ab By convention, time starts at 0. Furthermore, the finish time of a task equals its start time plus its duration:
$(\forall t : t \in \mathcal{T}_1 : \tau_{S_0}(t) \geq 0 \wedge \tau_{F_0}(t) = \tau_{S_0}(t) + \tau_{t_0}(t))$

C-Ac For consecutive tasks, it holds that the start time of the succeeding task is at least the finish time of the preceding task:
$(\forall t, t' : (t, t') \in P_1 : \tau_{S_0}(t') \geq \tau_{F_0}(t))$

C-Ad To match the states of consecutive tasks on the same resource, state transitions of the resource might be necessary. In these cases it holds that the start time of the succeeding task is at least the finish time of the preceding task plus the duration of the resource state transition between the tasks:
$(\forall t, t', r : (t, t') \in P_1', r \in I_1(t) \cap I_1(t') : \tau_{S_0}(t') \geq \tau_{F_0}(t) + \tau_{r_0}(r, Se_1(t, r), Sb_1(t', r)))$

where:

- $P_1' \subseteq P_1$ is the union of all resource task chains.
- $\tau_{t_0} : \mathcal{T}_1 \rightarrow \mathbb{R}^+$ gives the duration of a certain task, taking into account the behavioral restrictions imposed by the task as well as the resources involved with the task.
- $\tau_{r_0} : \mathcal{R} \times \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^+$ gives the duration of a resource state transition from some state to another state, taking into account the behavioral restrictions imposed by the resource.

  For further information on τ_{t_0} and τ_{r_0} see [11].

2.2.3 Instantiating the dual-stage wafer scanner scheduling problem

In the wafer scanner example, and in complex manufacturing machines in general, several selection alternatives exist. First of all, alternatives exist concerning precedences, which can be defined like in the Job Shop Scheduling problem. The precedence alternatives are outlined by P_2 together with constraint C-Ba assuring mutual exclusiveness. The additional precedence instances $P_1 \backslash P_2$ concern the scheduled interweaving of the instances that are the result of selection B.

Second, alternatives exist concerning involved resources. In some cases, multiple resources of the same kind are present in a machine, like the two wafer stages and the two elevators. As a consequence, several resources can be chosen from to allocate to a certain task. This is also the case in the generalized Job Shop Scheduling problem. For each type of resource, a capability is introduced in model element $\mathcal{C}$, e.g. 'wafer stage' and 'elevator'. The availability function $A: \mathcal{C} \rightarrow \mathcal{P}(\mathcal{R})$ describes which resources are available for a certain capability. For the example in Fig. 2.2 this function defines, e.g., that resources 'elv:0' and 'elv:1' are available for the 'elevator' capability. The resource involvement selection concerns selection of one available resource for each involved capability: constraint C-Bb.

Finally, alternatives exist concerning tasks. This is implied by the fact that in some cases multiple tasks exist in a system that have an equivalent effect considering the manufacturing process. For instance, multiple paths to transport material from one place to another, or a set of (m) tasks of which only a subset (n) has to be selected. This degree of freedom cannot be mapped onto the (generalized) job shop scheduling problem. Therefore, in the sequel of this section, analysis is done to determine which expressivity is required.

In Fig. 2.3, an overview of the alternatives with respect to tasks is given for the example case. Regarding the exposure scanning of dies, a 1 out of 2 expressivity is required: a die can be exposed (scanned) in two directions, of which one must be selected. This is analogous to the Rural Postman Problem: n out of m, in which $n = 1$, and $m = 2$ [9]. Regarding the measuring of mark pairs on a wafer, the requirements are more difficult. From the (m) mark pairs on a wafer, a minimum number (n) must be measured. A mark pair consists of two marks, each requiring a measure scan task in either direction. In this case the task selection can be defined as a selection out of m,

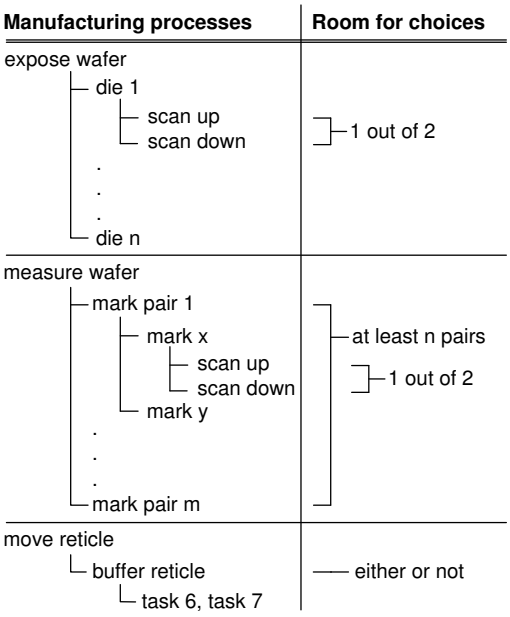


Figure 2.3: Alternatives with respect to tasks

in which the number of selected mark pairs must be an allowed number, and in which for all marks in the selected mark pairs one scan task is selected. From this, it can be concluded that nesting is needed and that the allowed number of alternatives can be more than one number ($n \in \mathcal{P}(\mathbb{N}^+)$). Furthermore, a complex machine contains buffer places like the IRL. At certain points in the manufacturing process, it is possible to buffer a manufacturing entity. To define this possibility in an intuitive way, it must be possible to describe the fact that also no buffering is allowed, or: 0 can also be an allowed number ($n \in \mathcal{P}(\mathbb{N})$). The required expressivity that follows from this analysis has not been found in literature.

2.3 Problem definition

To outline the required room for alternatives with respect to tasks, literature cannot be followed. In general, tasks and their precedence relation can be visualized by a graph of type 'activity on node'. For the purpose of modelling selection alternatives concerning tasks, a more general *node* element $\mathcal{N}_2$ is introduced. Equivalent alternatives can consist of a set of nodes (nesting). To identify such sets of nodes that all must be done and can have precedence edges to each other, an additional node type *cluster* ($\mathcal{L}_2$) is introduced. Function Ln_2: $\mathcal{L}_2 \rightarrow \mathcal{P}(\mathcal{N}_2)$ is introduced to define which nodes belong to which cluster. Furthermore, it is possible that multiple numbers of alternative nodes are allowed to be selected, including the possibility to select none of them. To be able to define which nodes belong to such a group of alternatives, a node type *group* is introduced: $\mathcal{G}_2$. Function Gn_2 is introduced to define which nodes are in which group, whereas function Ga_2 is introduced to define how many of these nodes are allowed to be selected. In Fig. 2.4, the node types are indicated for the example, and for groups the allowed numbers of selected alternatives is shown.

The newly introduced definition elements, together with some additional selection constraints outline the room for selections with respect to tasks. The resulting model can

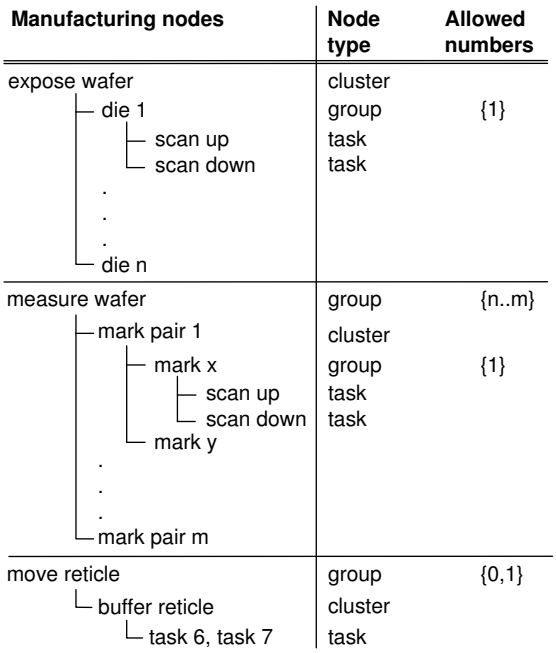


Figure 2.4: Alternatives with respect to tasks: node info

express $[n_1$ out of $m]$ or $[n_2$ out of $m]$, ... (aggregates of) tasks, which is abbreviated to $[\{n_1, n_2, ...n_x\}$ out of $m]$.

Summarizing this analysis, an instantiated, unselected TRS $\mathcal{D}_2$ can be defined by a 14–tuple $(\mathcal{T}_2, \mathcal{L}_2, \mathcal{G}_2, \mathcal{N}_2, \mathcal{R}, \mathcal{C}, I_2, A, P_2, Ln_2, Gn_2, Ga_2, Sb_2, Se_2)$:

- $\mathcal{T}_2$ is a finite set of elements called tasks.
- $\mathcal{L}_2$ is a finite set of elements called clusters.
- $\mathcal{G}_2$ is a finite set of elements called groups.
- $\mathcal{N}_2$ is a finite set of elements called nodes and is a generalization of the model elements mentioned earlier: $\mathcal{N}_2 = \mathcal{T}_2 \cup \mathcal{L}_2 \cup \mathcal{G}_2$.
- $\mathcal{R}$ is a finite set of elements called resources.
- $\mathcal{C}$ is a finite set of elements called capabilities.
- I_2: $\mathcal{T}_2 \rightarrow \mathcal{P}(\mathcal{C})$ gives the set of capabilities that are involved with a certain task.
- A: $\mathcal{C} \rightarrow \mathcal{P}(\mathcal{R})$ gives the set of resources that are available for a certain capability.
- $P_2 \subseteq \mathcal{N}_2 \times \mathcal{N}_2$ is the precedence relation between nodes.
- Ln_2: $\mathcal{L}_2 \rightarrow \mathcal{P}(\mathcal{N}_2)$ gives the set of nodes that are in a certain cluster.
- Gn_2: $\mathcal{G}_2 \rightarrow \mathcal{P}(\mathcal{N}_2)$ gives the set of nodes (alternatives) that a group consists of.
- Ga_2: $\mathcal{G}_2 \rightarrow \mathcal{P}(\mathbb{N})$ gives the allowed numbers (including 0) of nodes to be selected from a group.
- $Sb_2, Se_2 : TC \rightarrow \mathcal{S}$ give the begin and the end (physical) state of each capability involved in a certain task, where $TC = \{(t, c) | t \in \mathcal{T}_2, c \in I_2(t)\}$.

Constraint C-2a remains, and the additional constraints that have to be satisfied for the instances of the model elements are as follows:

C-2b There is no group that has only 0 as allowed number of selected nodes:
$(\nexists g : g \in \mathcal{G}_2 : Gn_2(g) = \{0\})$.

C-2c Nodes which are element of a group have no preceding or succeeding nodes:
$(\forall g, n : g \in \mathcal{G}_2, n \in Gn_2(g) : (\nexists n' : n' \in \mathcal{N}_2 : (n', n) \in P_2 \vee (n, n') \in P_2))$.

C-2d Precedences do not cross group boundaries.

By making choice B, a TRS definition of level 2 is transformed to a TRS definition of level 1. One of the choices has to do with selection from alternatives with respect to tasks. We define a node to be selected if at least one of the tasks that is in it is selected. Let $\mathcal{N}_1$ be the set of selected nodes, then the additional constraints for $f_B(\mathcal{D}_2)$ besides C-Ba and C-Bb can be formulated as follows:

C-Bc Precedence relations are inherited:
$(\forall n, n', t, t' : n, n' \in \mathcal{N}_1, t, t' \in \mathcal{T}_1, n \in anc(t), n' \in anc(t'), (n', n) \in P_2 : (t', t) \in P_1)$,
where function $anc : \mathcal{N}_2 \rightarrow \mathcal{P}(\mathcal{N}_2)$ is a recursive function that determines the ancestors of a node, which are those nodes in which a node n is contained:

$$anc(n) = (\cup n' : n' \in \mathcal{N}_2 \backslash \mathcal{T}_2, n \in Gn_2(n') \cup Ln_2(n') : \{n'\} \cup anc(n'))$$

This recursion is finite as the nodes have a hierarchical structure, which is explored upwards only in this function.

C-Bd Any node that is not part of another node must be selected, except for groups for which choosing nothing is allowed ('nilgroups'):
$(\forall n, l, g$: $n \in \mathcal{N}_2, l \in \mathcal{L}_2, g \in \mathcal{G}_2, n \notin Ln_2(l), n \notin Gn_2(g)$
$, n \notin \{g' \in \mathcal{G}_2 | 0 \in Ga_2(g')\} : n \in \mathcal{N}_1)$

C-Be Any node that is part of a selected cluster must be selected, except for nilgroups:
$(\forall l : l \in \mathcal{L}_2 \cap \mathcal{N}_1 : \{n \in Ln_2(l) | n \notin \{g' \in \mathcal{G}_2 | 0 \in Ga_2(g')\}\} \subseteq \mathcal{N}_1)$.

C-Bf In case nilgroups are not considered, the constraint for groups would be that the number of selected nodes that are part of a selected group must be allowed:
$(\forall g : g \in \mathcal{G}_2 \cap \mathcal{N}_1 : \#(Gn_2(g) \cap \mathcal{N}_1) \in Ga_2(g))$.
Presence of unselected nilgroups in a group relaxes this constraint somewhat, as unselected nilgroups may either or not be counted regarding the allowed number of selected nodes in a group. Knowing this, the constraint can be described as follows:
$(\exists a : a \in Ga_2(g) : \#(Gn_2(g) \cap \mathcal{N}_1) \leq a \leq$
$\#(Gn_2(g) \cap \mathcal{N}_1) + \#\{g' \in \mathcal{G}_2 \cap Gn_2(g) | 0 \in Ga_2(g') \wedge g' \notin \mathcal{N}_1\})$

Transformation A is not affected by the extension for choices with respect to tasks. To complete the formal definition of the optimization problem, a goal function $f_g : \mathcal{D}_0 \rightarrow \mathbb{R}$ is defined, that quantifies the quality of a certain temporal behavior. Examples of factors that play a role in this function are make span and number of tasks. Let $f_{AB} : \mathcal{D}_2 \rightarrow \mathcal{D}_0$ be the function that performs transformations A and B on the unselected TRS definition and returns the temporal machine behavior. The constraints that have to be satisfied for $f_{AB}(\mathcal{D}_2)$ can be constructed by combining the constraints of the separate transformations A and B: $f_{AB}(\mathcal{D}_2) = f_A(f_B(\mathcal{D}_2))$. Furthermore, two sets of valid functions for f_A and f_B are introduced, F_A, and F_B, respectively. With this, the entire optimization problem can be described as follows:

$$(\max f_A, f_B : f_A \in F_A, f_B \in F_B : f_g(f_A(f_B(\mathcal{D}_2))) \tag{2.1}$$

2.4 A scheduling algorithm

In this section, the optimization approach is discussed. First, the argumentation that has led to the approach and an outline of the algorithm are given. After that, the essential steps of the algorithm are explained.

The run-time usability requirement has important consequences. To avoid the machine being idle while waiting for its controller computing 'optimal' schedules, the algorithm is developed such that tasks can be dispatched to start execution with very small time delays. To achieve this, the schedule is determined in a constructive way, which means from the start to the finish. This approach is also safe with respect to extendability towards handling a TRS definition of level 3. Furthermore, it is possible to dispatch a partial schedule after each task that is added to it. To ensure that the dispatched schedule is an acceptable one, heuristic filters are used to direct the scheduling choices involving choice B. Note that if the algorithm is interrupted to dispatch the schedule, sub-optimal schedule solutions are taken for granted to just get the machine to work, and non-repetitive behavior might result as a consequence. Moreover, heuristic filters can be configured such that behavior of a state-based control architecture is copied, which is convenient for software migration purposes. Concerning choice A, the duration of tasks and setup resource state transitions between tasks is determined using dedicated mathematical functions for efficiency and embedability in SMC. Furthermore, the default heuristic of the approach with respect to selection A is to schedule a selected task 'As Soon As Possible', resulting in an 'active' schedule. For memory efficiency, a compact data structure is applied to store the result of selection B that is also compatible with the constructive and ASAP scheduling heuristic of selection A: a heap of pieces [18]. A piece defines a selected task and the selected involved resources, whereas the sequence of pieces in the heap defines the selected precedence relation. Depending on the goal function a postprocessing step is done with respect to selection A to postpone some tasks in order to improve the schedule. Subsequently, other choices with respect to selection B are considered. Taking the run-time aspect into account, the approach explores other alternatives at the beginning of the schedule first, as these tasks will be dispatched first. In Fig. 2.5, the approach is depicted in a flow chart. Summarizing, the approach is a constraint-guided heuristic search algorithm [12] with the possibility to dispatch work early if desired.

Below, steps 1 and 3 are described in detail. Steps 2 and 4 are very case-dependent.

Step 1

During transformation functionality B, selected alternatives are stored in a heap of pieces. Piece p describes which task $t \in \mathcal{T}_2$ is selected, and the selected resources $rr \subseteq \mathcal{R}$ involved: $p \in \mathcal{T}_2 \times \mathcal{P}(\mathcal{R})$. The sequence of pieces in the heap $h \in \mathcal{P}((\mathcal{T}_2 \times \mathcal{P}(\mathcal{R}))^*)$ describes the selected precedence relations in addition to the precedence relations in P_2 (intrinsically satisfying constraint C-Ba). The functions in this section are defined in the context of the system definition, therefore the definition elements are not explicitly included in the arguments.

If no alternatives with respect to tasks are taken into account, considering a heap h containing passed (i.e. selected up to then) tasks $t_p \subseteq \mathcal{T}_2$, a next piece p consisting of task t and involved resources rr is eligible to form an extended heap hp if and only if:

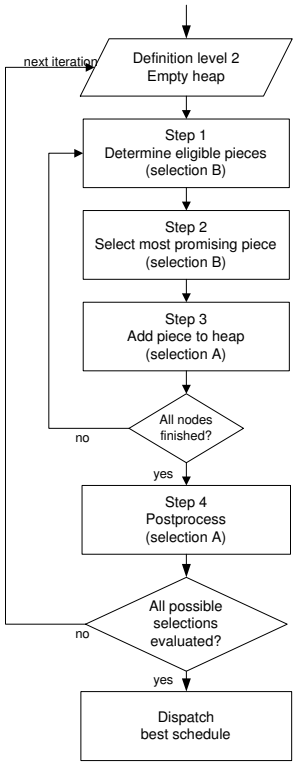


Figure 2.5: Flow chart of optimization approach

- the predecessors of t are in pieces of heap h, and t is not:

$$Et(h) = \{t \in \mathcal{T}_2 \backslash t_p \mid (\forall t' : (t', t) \in P_2 : t' \in t_p)\} \tag{2.2}$$

- constraint C-Bb is satisfied concerning rr:

$$Er(t) = \{rr \subseteq \mathcal{R} \mid (\forall r, c : r \in rr, c \in I_2(t) : r \in A(c))\} \tag{2.3}$$

The set of eligible next pieces for sequence h can be defined as follows:

$$E(h) = \{(t, rr) | t \in Et(h) \wedge rr \in Er(t)\} \tag{2.4}$$

The set of feasible heaps $\mathcal{H} \subseteq \mathcal{P}((\mathcal{T}_2 \times \mathcal{P}(\mathcal{R}_2))^*)$ can be defined by induction as follows:

$$\varepsilon \in \mathcal{H} \tag{2.5}$$

$$h \in \mathcal{H} \wedge p \in E(h) \Longrightarrow hp \in \mathcal{H} \tag{2.6}$$

In (2.5), ε denotes the empty heap.

Considering alternatives with respect to tasks, function $Et(h)$ must be extended. During the selection process, tasks that are not selected and will not be selected anymore are called 'bypassed'. After the selection process, the set of bypassed tasks equals $\mathcal{T}_2 \backslash \mathcal{T}_1$. Function $Et(h)$ needs to consider only tasks that are neither passed nor bypassed. For the tasks that are neither passed nor bypassed, the predecessor relation must be checked. In case no alternatives with respect to tasks are considered, all predecessors must be in the heap (see Equation 2.2). This condition is relaxed in case of predecessors of type group: all (possibly inherited, see constraint C-Bc) predecessors must be 'succeedable'.

Let function $succ : \mathcal{N}_2 \rightarrow \mathcal{P}(\mathcal{N}_2)$ be a function that determines the successors of a node n:

$$succ(n) = (\cup n' : n' \in \mathcal{N}_2, (\exists n'' : n'' \in anc(n) \cup \{n\} : (n'', n') \in P_2) : \{n'\}) \tag{2.7}$$

A task is succeedable when it is passed, and a cluster is succeedable when all nodes in it are succeedable. The non-succeedable nodes of a succeedable group may contain no passed tasks. Furthermore, if none of the nodes of a group is succeedable whereas zero is an allowed number, a group is succeedable when all of its predecessors are succeedable. In other cases, a group is succeedable when the (non zero) number of succeedable nodes of it is an allowed number.

Let $ns : \mathcal{N}_2 \rightarrow \mathbb{B}$ be a recursive function that determines whether a node n is succeedable.

$$\begin{aligned} ns(n) = \ & (n \in \mathcal{T}_2 \wedge n \in t_p) \\ & \vee (n \in \mathcal{L}_2 \wedge (\forall n' : n' \in Ln_2(n) : ns(n'))) \\ & \vee (n \in \mathcal{G}_2 \wedge (\forall n' : n' \in Gn_2(n) \wedge \neg ns(n') : (\forall t : t \in \mathcal{T}_2 \wedge n \in anc(t) : t \notin t_p)) \\ & \quad \wedge ((\nexists n' : n' \in Gn_2(n) : ns(n')) \wedge 0 \in Ga_2(n)) \\ & \qquad \wedge (\forall n' : n' \in \mathcal{N}_2 \wedge (n', n) \in P_2 : ns(n')) \\ & \qquad) \\ & \quad \vee (((\exists n' : n' \in Gn_2(n) : ns(n')) \vee 0 \notin Ga_2(n)) \\ & \qquad \wedge (|\{n' | n' \in Gn_2(n) \wedge ns(n')\}| \in Ga_2(n))) \\ & \qquad) \\ & \quad) \end{aligned} \tag{2.8}$$

This recursion is finite as the nodes have a hierarchical structure which is explored downwards only, and precedences have no loops and are explored backwards only. The set of succeedable nodes, n_s, can be defined as follows: $n_s = \{n | n \in \mathcal{N}_2 \wedge ns(n)\}$.

Let n_i be the set of initiated nodes. A node is initiated if it is not succeedable and contains a passed task. This set can be defined as follows:

$$n_i = (\cup n : n \in \mathcal{N}_2 \backslash n_s, (\exists t : t \in t_p : n \in anc(t)) : \{n\}) \tag{2.9}$$

A task is bypassed when it is not passed and when it is in a (node of a) group that is not succeedable or initiated whereas the maximum number of nodes of the group is succeedable or initiated, or if any succeeding node of it is succeedable. The set of bypassed tasks, t_b, is defined as follows:

$$\begin{aligned} t_b = & \{ t \in \mathcal{T}_2 \backslash t_p \\ & \quad | \quad (\exists g : g \in \mathcal{G}_2 : Gn_2(g) \cap (anc(t) \cup \{t\}) \backslash (n_s \cup n_i) \neq \emptyset \\ & \qquad \qquad \wedge |Gn_2(g) \cap (n_s \cup n_i)| = \max(Ga_2(g)) \\ & \qquad) \\ & \quad \vee (\exists t' \in t_p : (\{t\} \cup anc(t')) \cap succnil(t) \neq \emptyset) \\ & \quad \} \end{aligned} \tag{2.10}$$

Where function $succnil$: $\mathcal{T}_2 \rightarrow \mathcal{P}(\mathcal{N}_2)$ determines the successors of a task t, including the successors of succeeding nilgroups:

$$succnil(t) = succ(t) \cup (\cup n : n \in (succ(t) \cap \mathcal{G}_2) \wedge 0 \in Gn_2(n) : succnil(n)) \tag{2.11}$$

Using this, function $Et(h)$ when considering alternatives with respect to tasks is defined as follows:

$$Et(h) = \{t \in \mathcal{T}_2 \backslash t_p \ \backslash t_b | \ (\forall n, n' : n \in anc(t) \wedge (n', n) \in P_2 : n' \in n_s)\} \tag{2.12}$$

Step 3

An 'As Soon As Possible' (ASAP) heuristic for the choice concerning timing can be associated with an intuitive interpretation, namely that of a heap of pieces [4, 18]. Timing behavior of a TRS can be visualized using a Gantt chart. When a Gantt chart is turned 90° counter-clockwise, the resource occupation by tasks can be interpreted as a heap of pieces $p \in \mathcal{T}_2 \times \mathcal{P}(\mathcal{R})$. The first element of this tuple, $p.0$, equals the considered task, whereas the second element, $p.1$, equals the resources involved with this task: $p = (t, I_1(t))$. Resources can be associated with the slots on the horizontal axis, whereas (task duration) time is represented on the vertical axis. Tasks are represented by rectangular pieces. The task duration τ_{t_0} is represented by the height of a rectangular, whereas the involved resources are represented by its 'width'. The 'ASAP' heuristic can be associated with pieces falling onto each other under the influence of 'gravity'. This corresponds to the mechanism of the Tetris or Brick game.

The *upper contour* of a heap is associated with the time until which the resources are occupied by the pieces in the heap. It is defined as the $\mathcal{R}$-dimensional row vector $u_H(h)$, where $u_H(h,r)$ is the height of the heap on slot r. The *upper contour state* is defined as the $\mathcal{R}$-dimensional row vector $u_{Hs}(h)$, where $u_{Hs}(h,r)$ is the (physical) state of resource r at time $u_H(h,r)$.

The horizontal ground convention (see constraint C-Ab), which can be associated with time starting at 0, yields:

$$u_H(\varepsilon) = (0, ..., 0) \tag{2.13}$$

The upper contour of heap hp that results after piling up a piece p on top of a heap h is equal to the finish time of task t for the resources that are occupied by t and equal to the upper contour of h for the other resources:

$$u_H(hp, r) = \begin{cases} \tau_{F_0}(p.0, h) & \text{if } r \in p.1 \\ u_H(h, r) & \text{if } r \notin p.1 \end{cases} \tag{2.14}$$

The upper contour state of heap hp that results after piling up a piece p on top of a heap h is equal to the end state of task t for the resources that are occupied by t and equal to the upper contour state of h for the other resources:

$$u_{Hs}(hp, r) = \begin{cases} Se_1(p.0, r) & \text{if } r \in p.1 \\ u_{Hs}(h, r) & \text{if } r \notin p.1 \end{cases} \tag{2.15}$$

The finish time of a task t is obtained by adding its duration to its start time (see constraint C-Ab):

$$\tau_{F_0}(t, h) = \tau_{S_0}(t, h) + \tau_{t_0}(t) \tag{2.16}$$

The start time of a task t associated with piece p being piled up on top of a heap h is influenced by two components: by its preceding tasks (see constraint C-Ac) on the one hand and by the setup state transitions of the involved resources (see constraint C-Ad) on the other. Note that this precedence constraint is an extension of [3]. It can be determined by taking the highest value of either the highest finish time of its preceding tasks, $\tau_{S_{0p}}(t, h)$, or the highest part of the upper contour of the heap h beneath the piece after any required state transitions of the involved resources, $\tau_{S_{0r}}(t, h)$:

$$\tau_{S_0}(t, h) = (\max(\tau_{S_{0p}}(t, h), \tau_{S_{0r}}(t, h))) \tag{2.17}$$

Regarding preceding tasks, the start time of task t equals the maximum finish time of the passed tasks that precede t:

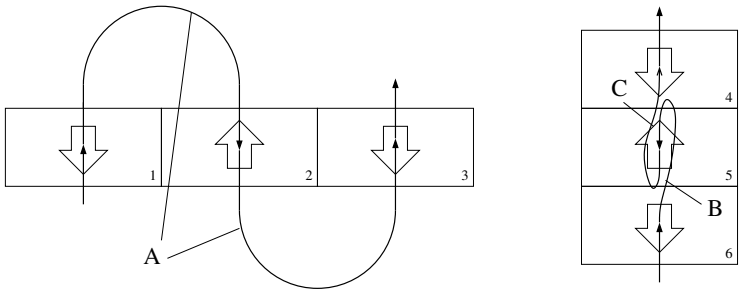


Figure 2.6: Horizontal and vertical meander

$$\tau_{S_{0p}}(t,h) = \ (\max t' : t' \in t_p \wedge t \in succnil(t') : \tau_{F_0}(t',h)) \tag{2.18}$$

Regarding resource setup, the start time of a task t can be obtained as follows:

$$\tau_{S_{0r}}(t,h) = (\max r : r \in I_1(t) : u_H(h,r) + \tau_{r_0}(r, u_{Hs}(h,r), Sb_1(t,r))) \tag{2.19}$$

2.5 Results

The results are illustrated using an example manufacturing scenario, concerning three lots of five wafers of a dual exposure recipe each. For each lot, different reticles are needed: reticles 0 and 1 for the first lot, reticles 2 and 3 for the second lot and reticles 4 and 5 for the third lot. Each wafer consists of 171 dies of size 26×13 [mm], in which the 13 [mm] is the scanning direction.

On each wafer 25 alignment mark pairs are placed, of which at least 16 pairs must be measured in either direction of each mark to reach the required minimum manufacturing accuracy.

For the purpose of this chapter, only selection of tasks and their order is considered. Alternatives can be found in the following areas:

1. The order and direction of exposing the dies of each wafer. Two basic approaches are shown in Fig. 2.6: a horizontal and a vertical meander.
2. The mark pairs to measure and the order of the measurements.
3. The reticles to buffer and the order of buffering.

For illustration, the timing behavior resulting from two different settings for the heuristic filters is shown. The description of the first setting, setting I, is as follows:

1. The exposure sequence is a horizontal meander, as visualized for this example in Fig. 2.7.
2. The minimum number (16) of mark pairs is measured.
3. Preprocessing of reticles is started as soon as possible. If reticles cannot go through to the reticle stage, they are put in the buffer (IRL).

In Fig. 2.8, the ASAP time behavior for setting I is depicted in a Gantt chart. The vertical axis shows the mechatronic systems that are distinguished as resources. In the

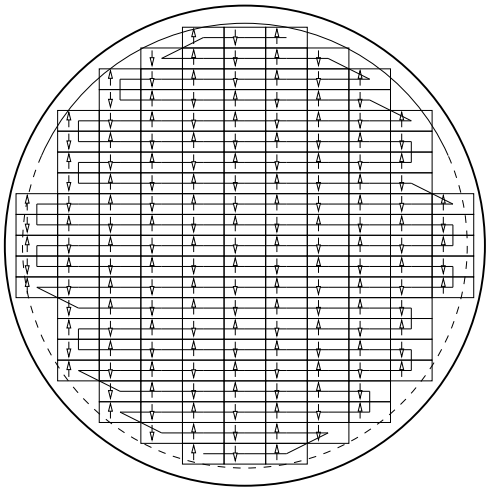


Figure 2.7: Horizontal meander for the example recipe

middle, there is the Light resource that is used for exposures, together with the reticle stage (RS) and one the wafer stages (Chu:0 and Chu:1). The resources depicted above the Light resource are reticle related, and can be found in Fig. 2.2. The resources depicted below the Light resource are wafer related. The Track is the machine next to the wafer scanner that delivers wafers to the pre-aligner (Prea), and takes them from the discharge unit (Dist). Furthermore, there are two wafer robots, one for the incoming path to the stages (Rob:i), and one for the outgoing path from the stages (Rob:o). In this chart, the measure and the exposure sequence are depicted as one task, and tasks are colored per wafer or reticle, where exposure tasks get the color of the reticle.

In Fig. 2.9, the critical path of the time behavior is depicted in a Gantt chart. The chart shows that inspection of reticles 3 and 4 is on the critical path, and exposure of the first lot is not.

The description of the second heuristic setting, setting II, is as follows:

1. The exposure sequence is a vertical meander.
2. All mark pairs (25) are measured.
3. Preprocessing of reticles is started only if less than four preprocessed reticles are available for coming exposures. If the robot can choose to either put a reticle on the turret or start preprocessing a next reticle, it chooses to put the reticle on the turret. Reticles are put in the buffer (IRL) after preprocessing only if they have to wait and the coming lot requires another reticle that is not preprocessed yet.

In Fig. 2.10, the resulting time behavior using setting II is depicted in a Gantt chart. As can be seen in Fig. 2.11, the inspection of reticles 3 and 4 is not on the critical path anymore, whereas exposure of the first lot is. The time needed for manufacturing the three lots is decreased by more than 5% compared to setting I. Half of this reduction is caused by the changed exposure sequence. The duration of this sequence itself decreased by about 10%. The other half is caused by changed reticle handling. Moreover, better product quality is achieved as more mark pairs are measured. This does not cost any time, as measuring still is not on the critical path, that is shown in Fig. 2.11.

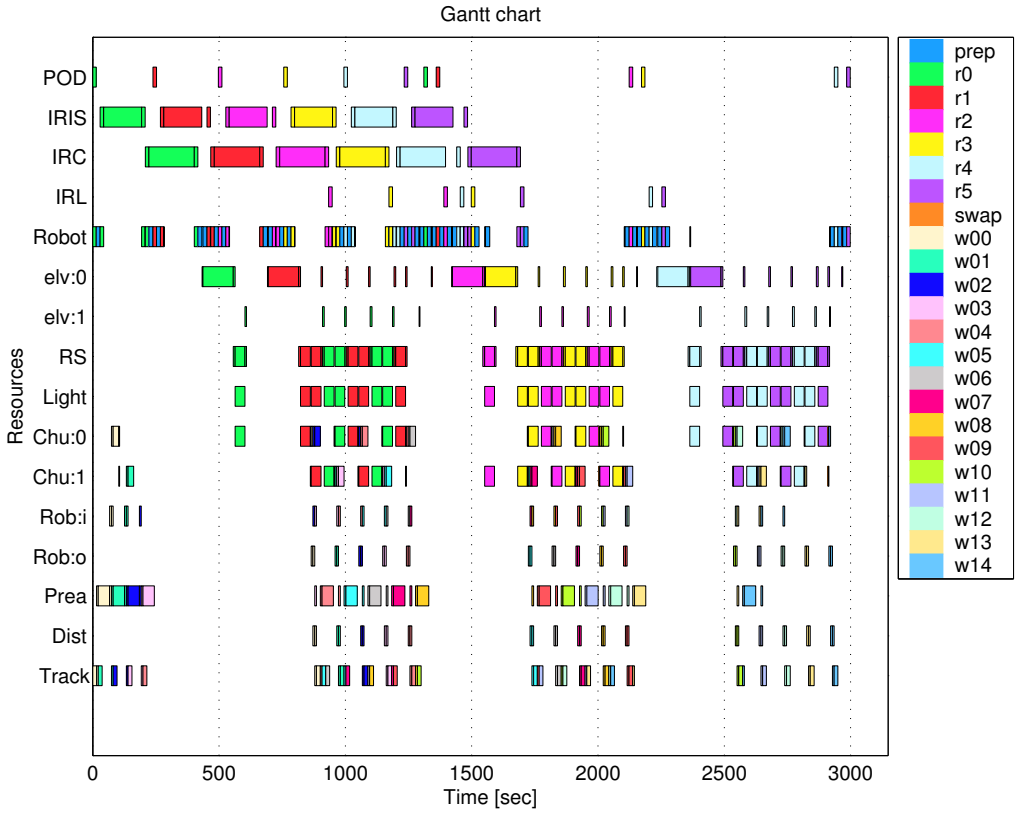


Figure 2.8: Gantt chart using heuristic setting I

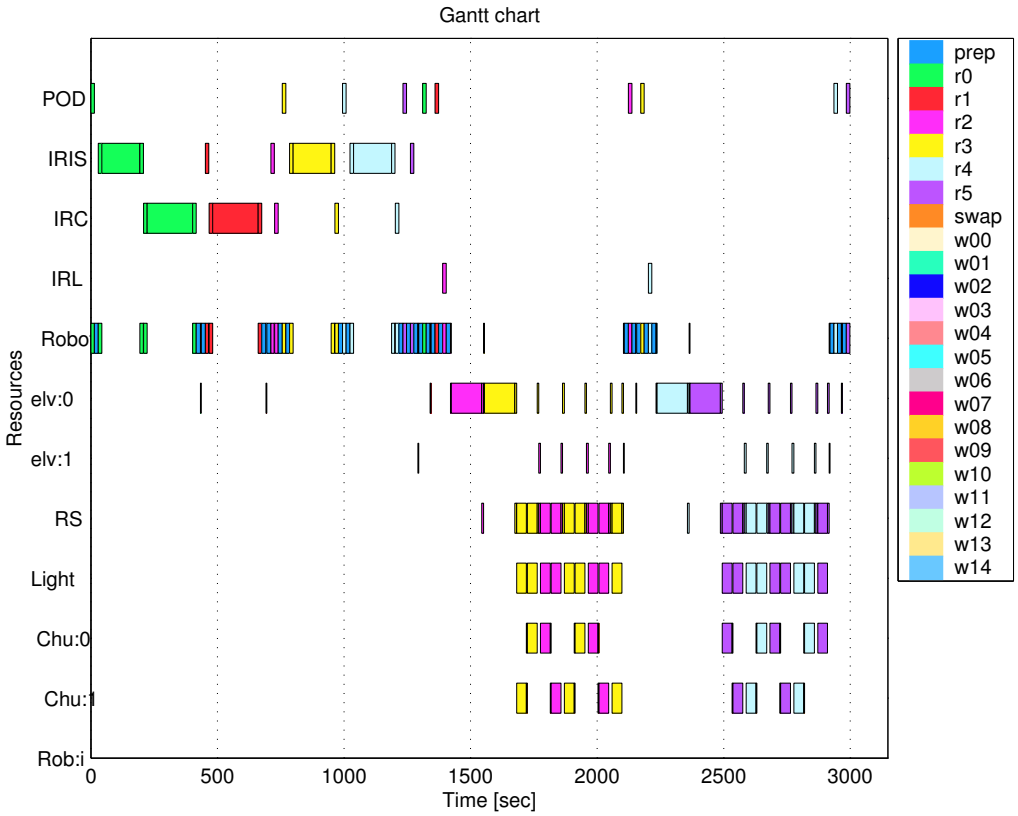


Figure 2.9: Gantt chart of critical path using heuristic setting I

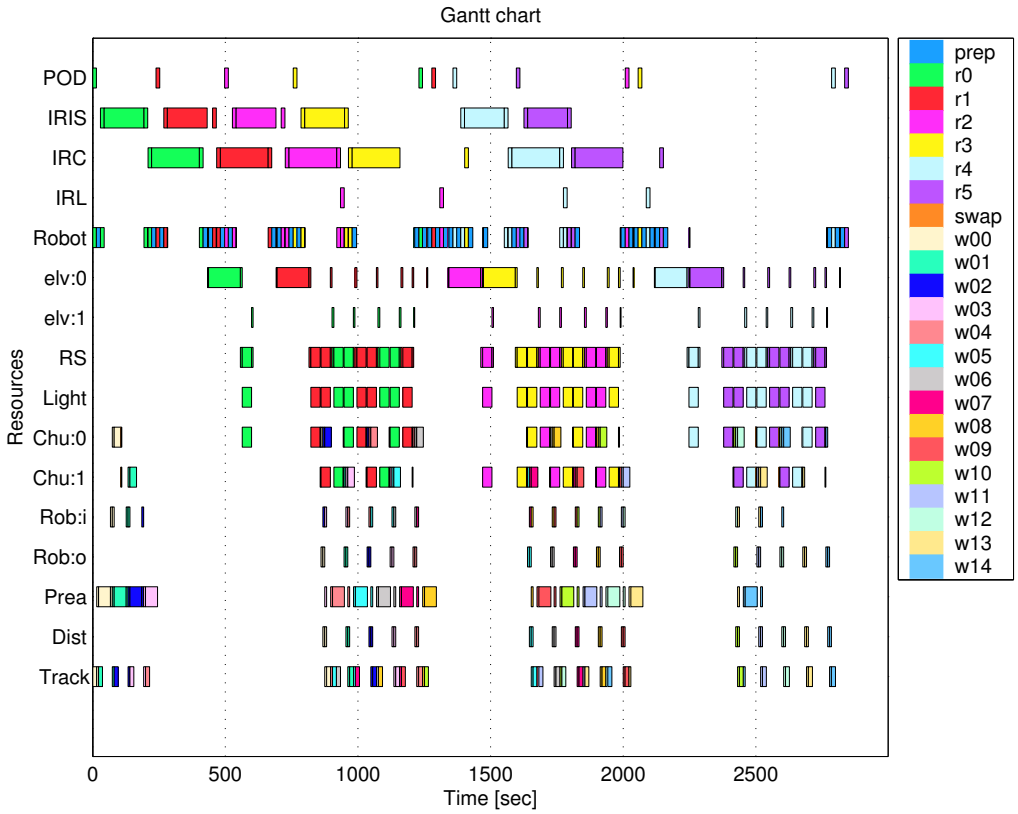


Figure 2.10: Gantt chart using heuristic setting II

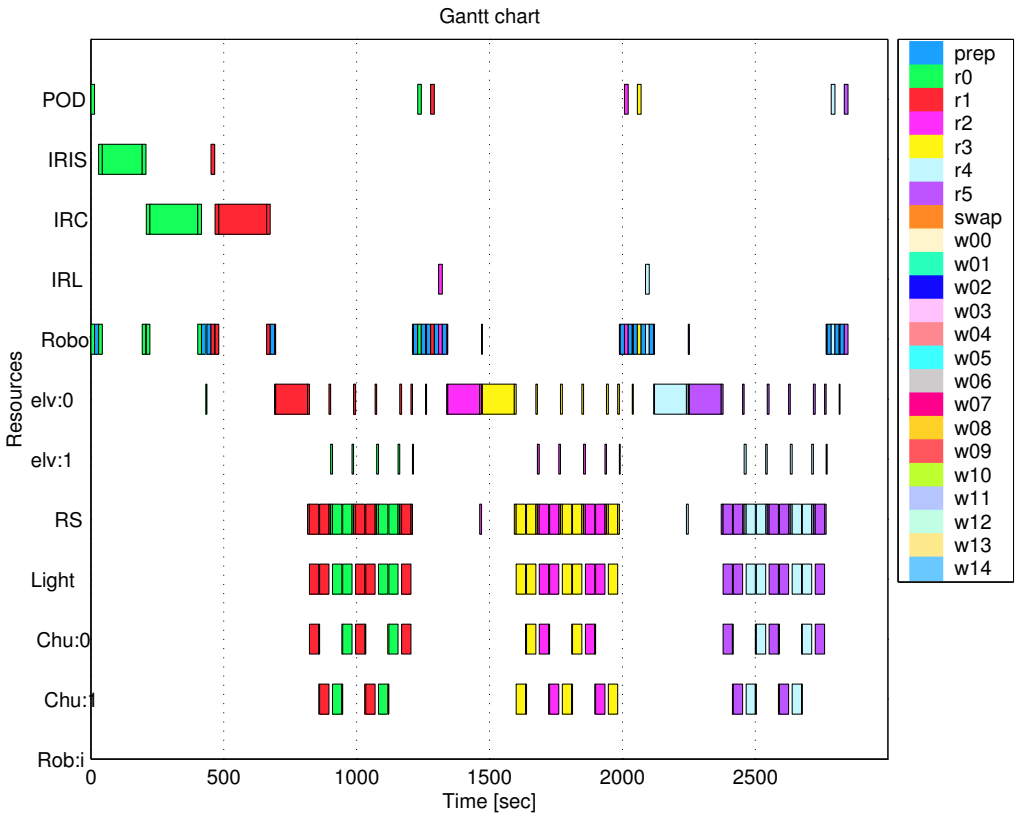


Figure 2.11: Gantt chart of critical path using heuristic setting II

2.6 Conclusions

Generalized job shop scheduling can form a basis for scheduling in complex machines. However, to account for choices with respect to tasks, extension of the job shop scheduling model is necessary. The scheduling problem in complex manufacturing machines is formally defined in the context of the layered TRS framework presented in [11]. Furthermore, a constraint-guided heuristic search scheduling algorithm is described that implements the selecting and timing transformations in the layered framework. It is suited for usage in the run-time environment of supervisory machine control as it features partial dispatching and uses a compact data structure.

Defining manufacturing recipes using graphs is intuitive and offers great expressivity concerning recipes, including the definition of alternatives with respect to tasks. The approach is flexible for strongly recipe dependent products and covers the entire machine (all tasks and resources, steady-state and transient behavior). Within the constraints introduced and the available scheduling time, real-time optimization of machine behavior is possible. The algorithm combines good behavior quality with little control overhead. In extremely time-critical situations, partial schedules consist of only one task: no prediction. In this case, the approach is similar to (current) state-based supervisory control [13, 15]. However, these approaches are not flexible for handling multiple product types at the same time. The applicability of the instantiated, unselected TRS and the possible behavior improvement is illustrated using an example from a wafer scanner. Results show that the definition elements allow intuitive modelling of the scheduling alternatives and that the scheduling algorithm allows for effective machine behavior optimization.

The following open issues remain. Unlike in a job shop, the restricted physical space in a complex manufacturing machine imposes additional requirements on valid schedules, e.g. to overcome overloading of resources or interference of resources. Such additional selection constraints implied by machine specific issues must be added [11]. Moreover, it must be verified that no deadlocks or other invalid behavior can be implied by the applied heuristic filters. Finally, the instantiation functionality (C) is to be developed to be able to react on triggers like manufacturing orders and exceptions by instantiating unselected TRS definitions. These open issues are subject of current research.

Acknowledgments

The authors would like to acknowledge Cor Hurkens for his valuable comments and Roel Boumen and Maarten van Bree for their help with the case.

References

[1] ASML, 2004. Information on wafer scanners available through URL `http://www.asml.com/`, item: products - lithography.

[2] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–341, 1994.

[3] S. Gaubert and J. Mairesse. Task resource models and (max,+) automata. In J. Gunawardena, editor, *Idempotency*, pages 131–144. Cambridge University Press, Cambridge, UK, 1998.

[4] S. Gaubert and J. Mairesse. Modeling and analysis of timed Petri nets using heaps of pieces. *IEEE Transactions on Automatic Control*, 44(4):683–697, 1999.

[5] P. Gohari and W. M. Wonham. Reduced supervisors for timed discrete-event systems. *IEEE Transactions on Automatic Control*, 48(7):1187–1198, 2003.

[6] D. Jevtic. Method and apparatus for automatically generating schedules for wafer processing within a multichamber semiconductor wafer processing tool, 1997. Patent no. US 6,201,999.

[7] J. Kim, T. Lee, H. Lee, and D. Park. Scheduling analysis of time-constrained dual-armed cluster tools. *IEEE Transactions on Semiconductor Manufacturing*, 16(3):521–534, 2002.

[8] S. Kumar, N. Ramanan, and C. Sriskandarajah. Robotic system control, 2003. Patent no. US 6,556,893.

[9] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley-Interscience, Chichester, 1985.

[10] H. Marchand, O. Boivineau, and S. Lafortune. On the synthesis of optimal schedulers in discrete-event control problems with multiple goals. *SIAM Journal on Control Optimization*, 39(2):512–532, 2000.

[11] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, and J. E. Rooda. Design of supervisory machine control. In K. Glover and J. Maciejowski, editors, *Proceedings of the European Control Conference 2003*, 2003. CD-ROM.

[12] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, 1995.

[13] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[14] S. Rostami and B. Hamidzadeh. Optimal scheduling techniques for cluster tools with process-module and transport-module residency contraints. *IEEE Transactions on Semiconductor Manufacturing*, 15(3):341–349, 2002.

[15] Y. Shin, T. Lee, J. Kim, and H. Lee. Modeling and implementing a real-time scheduler for dual-armed cluster tools. *Computers in Industry*, (45):13–27, 2001.

[16] S. F. Smith. Is scheduling a solved problem? In G. Kendall, E. Burke, and S. Petrovic, editors, *Multidisciplinary International Conference on Scheduling : Theory and Applications(MISTA'03)*, pages 11–20. ASAP, University of Nottingham, UK, August 2003.

[17] G. E. Vieira, J. W. Herrmann, and E. Lin. Rescheduling manufacturing systems: a framework of strategies, policies and methods. *Journal of scheduling*, 6(1):35–58, 2003.

[18] G. X. Viennot. Heaps of Pieces, I: Basic definitions and combinatorial lemmas. In G. Labelle and P. Leroux, editors, *Combinatoire Enumerative*, pages 321–350. Springer, New York, 1986.

[19] M. Wennink. *Algorithmic Support for Automated Planning Boards.* PhD thesis, Eindhoven University of Technology, The Netherlands, 1995.

CHAPTER

THREE

MACHINE-SPECIFIC SCHEDULING CONSTRAINTS

This chapter contains the paper *Predictive Scheduling in Complex Manufacturing Machines: Machine-Specific Scheduling Constraints* that has been protected in patent application ASML ref. P-1784. First filing was in the US at Dec 23, 2003, number 10/743,320. The paper was submitted to IEEE Transactions on Semiconductor Manufacturing in February 2004.

Predictive scheduling in complex manufacturing machines: machine-specific constraints

N.J.M. van den Nieuwelaar †*, J.M. van de Mortel-Fronczak †,
N.C.W.M. Braspenning †, J.E. Rooda †

Abstract

Supervisory control of a complex manufacturing machine - which involves coordination of many mechatronic systems - requires proper scheduling. Supervisory control must be flexible to concurrently process a mix of different product types each requiring heavily recipe-dependent manufacturing tasks, without introducing unnecessary control overhead. This chapter extends the generalized job shop scheduling model for the restrictions on the physical space inside complex manufacturing machines. To formally describe the scheduling process that transforms the model into a certain behavior, the transformation is split into two phases: selecting and timing, according to a layered task resource system framework. Selecting involves selection of which tasks to do in which order by which resource, whereas timing involves assignment of start and finish times to tasks. First, selecting constraints are introduced that ensure physically feasible behavior concerning the material involved in the manufacturing process: material flow integrity, material flow feasibility, and material capacity feasibility. To minimize scheduling overhead, it must be possible to dispatch a partial schedule. To avoid any invalid behavior such as deadlocks, additional selecting constraints are introduced. Furthermore, timing constraints are introduced to ensure feasible behavior concerning resource interference. Throughout the chapter, the approach is illustrated using a wafer scanner example.

3.1 Introduction

The purpose of a manufacturing machine is to make products, which requires physical manufacturing processes to be carried out. To actually do the work, mechatronic systems in the machine must be deployed. Control in the separate mechatronic systems is referred to as low-level control and is not considered in this chapter. In complex manufacturing machines, many options exist to deploy the available resources to perform tasks that lead to the desired manufacturing purpose, resulting in various machine behaviors. Supervisory Machine Control (SMC) is responsible for deciding when to do which tasks using which resources. There are three important complicating requirements for SMC of complex manufacturing machines. First of all, the manufacturing tasks are heavily product recipe dependent, for which SMC must be flexible. Furthermore, it must be able to handle a stream of mixed product types, which are being processed concurrently. Finally, no unnecessary control overhead may be introduced.

† Eindhoven University of Technology: P.O. box 513, 5600 MB Eindhoven, The Netherlands.
* ASML: De Run 6501, 5504 DR Veldhoven, The Netherlands.
Corresponding author: N.J.M. van den Nieuwelaar, e-mail: n.j.m.v.d.nieuwelaar@tue.nl

3.1.1 Literature

Many approaches exist to describe a system under supervisory control using well-known formalisms from computer science. Supervisory control theory as discussed by Wonham et al. [2, 3, 11] models the system under control using Finite State Machines. The possible behavior of such a system is regarded as a language. A supervisory controller in the form of a deterministic automaton is synthesized that restricts the language by disabling a subset of events, to control the system to properly accomplish its task. Supervisors must be modelled specifically for the task to be accomplished, and therefore are not flexible for handling different recipes.

Literature on performance analysis and supervisory control of complex manufacturing machines that can handle different recipes [5, 12, 13] encapsulates models of the tasks to be done for different recipes, which can be associated with scheduling theory. However, in this work only part of the total problem is analyzed. Some of the work is restricted to steady state behavior. As in complex manufacturing machines the time spent for transient behavior (e.g. while switching product types) is of the same order of magnitude as steady-state behavior, it is of importance not to focus on steady-state behavior only. Other work is restricted with respect to other areas, for example a single resource [6] or a subset of tasks [4], whereas SMC is responsible for control of the entire machine. Moreover, the dynamic and real-time circumstances that SMC must operate in should get sufficient attention, which is often underexposed in scheduling literature [14]. To properly address the recipe-dependent tasks to be done, an approach based on a well-known scheduling problem is proposed in this chapter, as is also done in [7].

3.1.2 Layered task resource system framework

From a SMC point of view, a machine can be considered as a task resource system (TRS). Tasks can be associated with manufacturing processes, whereas resources can be associated with mechatronic systems. Transforming a manufacturing request into machine behavior can be structured in three phases. First, a scheduling problem must be instantiated from the manufacturing request, taking into account the limitations of the machine. This transformation is called *instantiating*. The structure of the resulting scheduling problem shows many similarities with the job shop scheduling problem [10]. The manufacturing process of a material instance can be associated with a job, whereas the different parallel mechatronic systems can be associated with the different machines in a job shop. Subsequently, resources must be assigned to the tasks in the instantiated scheduling problem in some order, taking into account the fact that resources are able to perform certain tasks only, and only one at a time. This transformation is called *selecting*. The selected order of tasks to be performed by selected resources implies consecutive state transitions of those resources, which is analogous to the setup times for mode switching in job shop scheduling. Finally, start and finish times can be assigned to the tasks, taking into account the speed of the resources. This transformation is called *timing*.

During the three transformation phases of instantiating, selecting and timing, choices must be made. The result of a choice in a certain transformation on the machine behavior can only be evaluated by performing the consecutive transformations. Therefore, a transformation phase strongly relies on information from subsequent phases. The layered TRS framework shown in Fig. 3.1 displays the hierarchically related transformation phases as functionality layers (A through C) and the different TRS definition levels (0 through 3) as interfaces between the layers (see Chapter 1 and [9]).

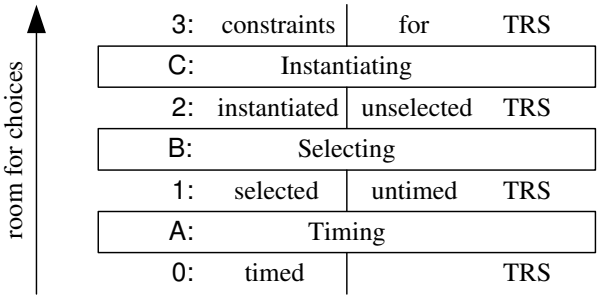


Figure 3.1: Layered Task Resource System framework

3.1.3 Predictive scheduling in Supervisory Machine Control

This chapter discusses predictive scheduling [15] in SMC based on task-resource system definitions. Scheduling can be associated with transforming a TRS definition of level 2 into a TRS definition of level 0, involving the layers A and B shown in Fig. 3.1. Several motivations exist for predictive scheduling in SMC of complex machines. First, making choices run-time enables optimized machine behavior for different products and users. Several optimization approaches can be applied: heuristics can be used for guidance and several schedules can be generated in several ways to do 'what-if' analysis. Second, the TRS definitions used are suited for design-time analysis and any error-prone gap between the design and the implementation is minimized as run-time execution is based on the same model. Finally, predictive scheduling is flexible, which eases adaptation to the evolution of machine configurations or machine operation philosophies. A potential drawback of predictive scheduling is control overhead, which slows down the machine. To minimize this effect, a constructive scheduling algorithm can be applied that allows for partial schedule dispatching. In any case, the schedules generated should be feasible and valid - which means that the machine should be able to carry out the schedule as scheduled, and that invalid behavior (such as deadlock) should be avoided.

3.1.4 Job shop scheduling

Although the scheduling problem in machines shows many similarities with the generalized job shop scheduling problem, there are some important differences. The most important difference is the restrictions on physical space. Whereas a job shop has plenty of room to store material and resources do not interfere with each other, this is not the case in a complex machine. Moreover, material transport time in a job shop is much less than processing time. Therefore, a job shop scheduling problem does not have a notion of material, and neglects material transport. The purpose of this chapter is to address the consequences of the tight physical space in a machine for the scheduling problem, as well as for a constructive scheduling algorithm. The problem definition is based on the job shop scheduling problem, and follows the framework shown in Fig. 3.1. To avoid infeasible machine behavior with respect to material, the notion of material and the constraints concerning material (which affect selecting, see B in Fig. 3.1) are introduced. Furthermore, constraints to avoid resource interference are introduced, which affect timing (see A in Fig. 3.1). This chapter also addresses how to avoid deadlocks during selecting to enable partial schedule dispatching.

3.1.5 Structure of the chapter

The structure of this chapter is as follows. Throughout the chapter, an example of a complex machine is used for illustration purposes: a dual-stage wafer scanner [1]. Other examples of complex manufacturing machines in the semiconductor industry are cluster tools and tracks. In Section 3.2, the wafer scanner example is described and from this a generalized job shop scheduling problem is instantiated. The machine-specific issues that do not map onto the job shop scheduling problem are pointed out. Section 3.3 describes the additional TRS definition level 2 elements and selecting constraints concerned with the notion of material and ensuring feasible machine behavior.

Furthermore, avoidance of invalid machine behavior such as deadlocks using additional selecting constraints is discussed. Section 3.4 discusses the additional TRS definition elements and timing constraints concerning resource interference. Both Sections 3.3 and 3.4 end with a constructive scheduling algorithm implementing the machine-specific scheduling constraints. In Section 3.5, the additional elements are instantiated for the example and a resulting schedule that indeed satisfies the machine-specific constraints is presented. Finally, concluding remarks are presented in Section 3.6.

3.2 Scheduling in a dual-stage wafer scanner

3.2.1 A dual-stage wafer scanner

The primary manufacturing process of a wafer scanner is the exposure of a mask containing an IC pattern onto wafers. A neighboring machine named 'track' performs some pre-processing and post-processing steps. As the required accuracy of the exposure process is very high, any imperfections concerning the wafers must be corrected for. To be able to do this, wafers are measured before being exposed. Both the measuring step and the exposure step take place at a wafer stage. The orientation of the wafer at a wafer stage is of importance for successful measurement and exposure, whereas the orientation is unknown when a wafer comes into the machine. Therefore, an alignment system is incorporated. Furthermore, the wafer scanner under consideration uses Extreme Ultra Violet light for exposure. As this light is absorbed by air, exposure must take place in a vacuum, whereas the machine is at atmospheric pressure. To bring the wafers from atmospheric pressure down to a vacuum, a load lock is incorporated. To transport the wafers between the different subsystems, a robot is used. A schematic layout of the wafer scanner is shown in Fig. 3.2. In this figure, circles depict the parallel mechatronic systems considered, and arrows depict the possible transport paths. Each mechatronic system can carry only one wafer, which is depicted between brackets. The tight layout of the machine makes it possible for the robots to collide if they both move from or to a lock, which is depicted by the double-dashed area. The wafer scanner is a dual-stage wafer scanner with a separate measure and expose area. In the measurement area, wafers can be loaded onto and unloaded from a stage at their load and unload positions, respectively. In general, resources can have an infinite number of states, for instance the coordinates of the wafer stages. For the purpose of this chapter, each resource can reach a limited number of states. The Finite State Machines (or automata) of the different mechatronic systems are shown in Fig. 3.3. In this figure, an extra circle denotes the initial state, and transitions are labelled with a time duration and possibly a task name, which will be explained later. The wafer stages, resources S0 and S1, can be in three states, corresponding with

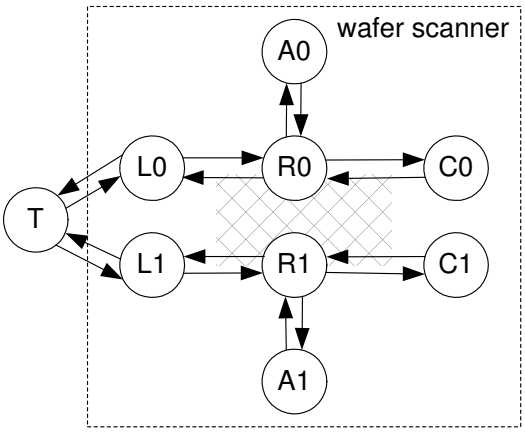


Figure 3.2: Schematic layout of a dual-stage wafer scanner

three locations: at the load or measure area (@lm), at the expose area (@e), or at the unload area (@u). Switching between areas of the stages must be done synchronously to avoid collision of the wafer stages: this is known as chuck swap. The dashed connections between the diagrams of the two stages depict this synchronism. The locks, resources L0 and L1, can be either at atmospheric pressure(atm) or at vacuum (vac). The track, resource T0, can be in three states: 'ready to send', 'ready to receive', and 'received a wafer'. The robots, resources R0 and R1, have three main states, corresponding with three locations: at the lock (@l), at the stage (@s), or at the alignment unit (@a). The states @ca and @cs model the limits of the collision-hazardous area. The manufacturing scenario used in this chapter concerns a typical batch (lot) of 15 wafers.

3.2.2 Job Shop Scheduling

In this subsection, job shop scheduling starting from an instantiated, unselected TRS definition (level 2 in Fig. 3.1) is formally defined. First, the instantiated, unselected TRS definition of a job shop scheduling problem is described, which subsequently is extended following generalized job shop scheduling. After that, the selected, untimed TRS definition is described, and the constraints to be taken into account when selecting from the alternatives defined in definition level 2 to reach definition level 1. Finally, the timed TRS definition (level 1) and the constraints concerning transformation A are summarized.

A job shop scheduling model can be defined by a 6-tuple in terms of tasks and resources $(\mathcal{T}_2, \mathcal{R}, I_2, P_2, Sb_2, Se_2)$ in which:

- $\mathcal{T}_2$ is a finite set of elements called tasks.
- $\mathcal{R}$ is a finite set of elements called resources.
- I_2: $\mathcal{T}_2 \rightarrow \mathcal{R}$ gives the resource that is involved in a certain task.
- $P_2 \subseteq \mathcal{T}_2 \times \mathcal{T}_2$ is the precedence relation between tasks.
- $Sb_2, Se_2 : TR \rightarrow \mathcal{S}$ give the begin and the end (physical) state of the resource involved in a certain task, where $TR = \{(t, r) | t \in \mathcal{T}_2, r \in I_2(t)\}$.

Note that, by convention, the definition level is added to each element in subscript. Elements that are not level-specific have no suffix.

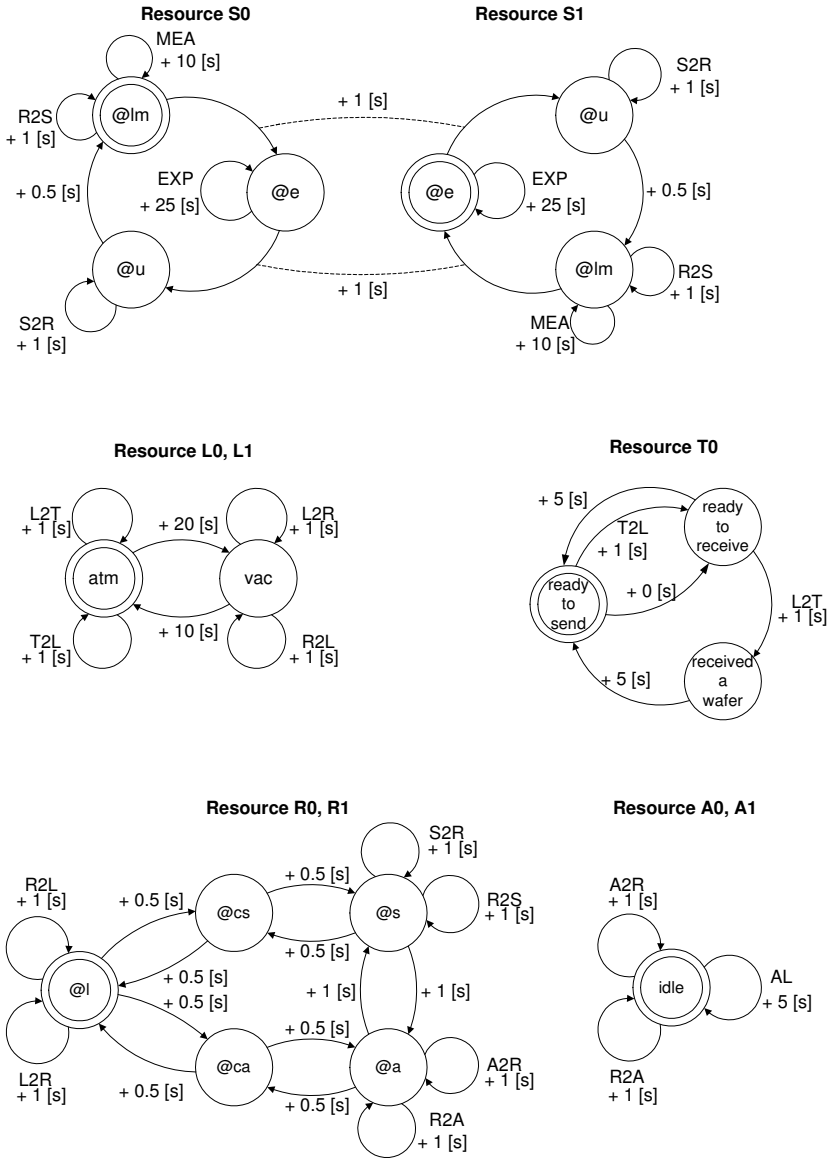


Figure 3.3: Resource automata

In the sequel, several constraints are defined and labeled according to the following convention. Each constraint label starts with a C- followed by the transformation layer identifier (A, B, C) or definition level identifier (0, 1, 2, 3) and subsequently by a letter (a, b, ...).

The constraint that must be satisfied for the instances of the definition elements is as follows:

C-2a P_2 contains no cycles.

However, like in the wafer scanner example, in a complex machine multiple resources can exist which all are capable of the same work. Furthermore, some tasks involve synchronous transitions of multiple resources, e.g. a scan. As this can also be the case in job shops, the job shop scheduling problem has been generalized to incorporate these features [16].

A generalized job shop scheduling problem can be defined by an 8–tuple $(\mathcal{T}_2, \mathcal{R}, \mathcal{C}, I_2, A, P_2, Sb_2, Se_2)$:

- $\mathcal{T}_2$ is a finite set of elements called tasks.
- $\mathcal{R}$ is a finite set of elements called resources.
- $\mathcal{C}$ is a finite set of elements called capabilities.
- I_2: $\mathcal{T}_2 \rightarrow \mathcal{P}(\mathcal{C})$ gives the set of capabilities that are involved in a certain task.
- A: $\mathcal{C} \rightarrow \mathcal{P}(\mathcal{R})$ gives the set of resources that are available for a certain capability.
- $P_2 \subseteq \mathcal{T}_2 \times \mathcal{T}_2$ is the precedence relation between tasks.
- $Sb_2, Se_2 : TC \rightarrow \mathcal{S}$ give the begin and the end (physical) state of each capability involved in a certain task, where $TC = \{(t, c) | t \in \mathcal{T}_2, c \in I_2(t)\}$.

Constraint C-2a remains.

The selecting transformation (B) assigns resources to tasks, and determines the order in which tasks are executed for each resource. By this transformation, an unselected TRS is transformed into a selected, untimed TRS, which can be defined by a 6-tuple $(\mathcal{T}_1, \mathcal{R}, I_1, P_1, Sb_1, Se_1)$:

- $\mathcal{T}_1$ is a finite set of elements called tasks.
- $\mathcal{R}$ is a finite set of elements called resources.
- I_1: $\mathcal{T}_1 \rightarrow \mathcal{P}(\mathcal{R})$ gives the set of resources that are involved in a certain task.
- $P_1 \subseteq \mathcal{T}_1 \times \mathcal{T}_1$ is the precedence relation between tasks.
- $Sb_1, Se_1 : TR \rightarrow \mathcal{S}$ give the begin and the end (physical) state of the resource involved in a certain task, where $TR = \{(t, r) | t \in \mathcal{T}_1, r \in I_1(t)\}$.

Constraints that have to be satisfied for the instances of the definition elements are as follows:

C-1a The sequence of tasks per resource is a chain.

Constraints that have to be satisfied for the selecting transformation can be formulated as follows:

C-Ba The sequence of selected tasks per resource is a chain (equals C-1a).

C-Bb For each selected task, an available resource must be selected for each involved capability:
$(\forall t, r, c : t \in \mathcal{T}_1, r \in I_1(t), c \in I_2(t) : r \in A(c))$

The begin and end state definition is obtained from the capability of the selected resource.

By the timing transformation (A), a selected, untimed TRS is transformed into a timed TRS, which can be defined by a 5-tuple $(\mathcal{T}_0, \mathcal{R}, I_0, \tau_{S_0}, \tau_{F_0})$:

- $\mathcal{T}_0$ is a finite set of elements called tasks.
- $\mathcal{R}$ is a finite set of elements called resources.

- I_0: $\mathcal{T}_0 \rightarrow \mathcal{P}(\mathcal{R})$ is the set of resources that are involved in a certain task.
- τ_{S_0}, τ_{F_0}: $\mathcal{T}_0 \rightarrow \mathbb{R}^+$ are the start time and the finish time of a certain task, which implies that all resources assigned to a task t are occupied for the same time span.

Furthermore, note that a timed TRS can be visualized as a Gantt chart.

The constraint that has to be satisfied for the instances of the definition elements is as follows:

C-0a Per resource there is a chronological sequence of pairs of task start and task finish times.

Constraints that have to be satisfied for the timing transformation are as follows:

C-Aa Nothing changes with respect to tasks and the (involved) resources: $\mathcal{T}_0 = \mathcal{T}_1, I_0 = I_1$

C-Ab By convention, time starts at 0. Furthermore, the finish time of a task equals its start time plus its duration:
$(\forall t : t \in \mathcal{T}_1 : \tau_{S_0}(t) \geq 0 \wedge \tau_{F_0}(t) = \tau_{S_0}(t) + \tau_{t_0}(t))$

C-Ac For consecutive tasks, it holds that the start time of the succeeding task is at least the finish time of the preceding task:
$(\forall t, t' : (t, t') \in P_1 : \tau_{S_0}(t') \geq \tau_{F_0}(t))$

C-Ad To match the states of consecutive tasks on the same resource, state transitions of the resource might be necessary. In these cases it holds that the start time of the succeeding task is at least the finish time of the preceding task plus the duration of the resource state transition between the tasks:
$(\forall t, t', r : (t, t') \in P_1', r \in I_1(t) \cap I_1(t') : \tau_{S_0}(t') \geq \tau_{F_0}(t) + \tau_{r_0}(r, Se_1(t, r), Sb_1(t', r)))$

where:

- $P_1' \subseteq P_1$ is the union of all resource task chains.
- $\tau_{t_0} : \mathcal{T}_1 \rightarrow \mathbb{R}^+$ gives the duration of a certain task, taking into account the behavioral restrictions imposed by the task as well as the resources involved with the task.
- $\tau_{r_0} : \mathcal{R} \times \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^+$ gives the duration of a resource state transition from some state to another state, taking into account the behavioral restrictions imposed by the resource.

  For further information on τ_{t_0} and τ_{r_0} see [9].

3.2.3 Instantiating the dual-stage wafer scanner scheduling problem

The mapping of the wafer scanner scheduling problem described earlier onto the definition of a generalized job shop scheduling problem can be split into two sections: system-dependent elements and work-dependent elements. The system-dependent elements can be defined as follows:

- There are five capabilities: stage, robot, alignment unit, lock and track:
  $\mathcal{C} = \{\text{S, R, A, L, T}\}$.
- There are nine resources: stage0, stage1, robot0, robot1, aligner0, aligner1, lock0, lock1, track0:
  $\mathcal{R} = \{\text{S0, S1, R0, R1, A0, A1, L0, L1, T0}\}$.

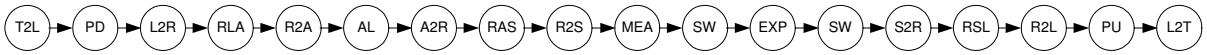


Figure 3.4: Steps in the manufacturing process of a wafer

Figure 3.5: Life of a wafer

- The available resources for each capability are defined as follows:
  A = {(S, {S0, S1}), (R, {R0, R1}), (A, {A0, A1}), (L, {L0, L1}), (T, {T0})}.

To define the work-dependent elements, the steps in the manufacturing process of a wafer are analyzed. First, a wafer is transported from the track into a lock (T2L). Subsequently, the pressure is pumped down (PD), and the wafer is transported onto the robot (L2R). The robot rotates from the lock to the alignment unit (RLA), places the wafer onto the alignment unit (R2A), and the alignment takes place (AL). After that, the robot takes the wafer from the alignment unit (A2R), rotates to the stage (RAS), and places the wafer onto the stage (R2S). On the stage, measurement takes place (MEA), and after stage swap (SW) the wafer is exposed (EXP). Then, the stage swaps to the unload position in the measure area (SW) where the robot takes the wafer from the stage (S2R). The robot rotates to the lock (RSL), and puts the wafer in the lock (R2L). Finally, the lock pumps up the pressure (PU) and the wafer is taken from the lock by the track (L2T). The steps in the manufacturing process of a wafer can be defined by $\mathcal{T}_2$ and P_2, and can graphically be displayed by a graph, as is shown in Fig. 3.4. A first attempt to define the task graph for the entire batch could be to define 15 of these identical sequences.

However, when looking more closely at the steps, it appears that some of the steps in Fig. 3.4 may be necessary besides the steps for each wafer. For example, if a lock subsequently has to pump down two wafers, it must pump up in between the two pump down steps. The same is true for the rotate steps and the stage swap. In job shop scheduling, such steps are called setups, which implies that they are a consequence of the sequence of regular tasks using the same resource. Depending on the selected sequence of tasks using the same resource, such steps may or may not be required. More specifically: if for some resource the end state of a preceding task does not match the begin state of a consecutive task, a state transition is inserted to bridge this gap. This implies that such state transitions can be left out of the task sequence or 'life of' a wafer, as is shown in Fig. 3.5.

For one wafer, for example wafer W1, the work-dependent elements can be instantiated as follows:

- $\mathcal{T}_2$ = {W1-T2L, W1-L2R, ...}.
- P_2 = {(W1-T2L, W1-L2R), (W1-L2R, W1-R2A), ...}.
- I_2 = {(W1-T2L, {T, L}), (W1-L2R, {L, R}), (W1-R2A, {R, A}), ...}.
- Sb_2 = {((W1-T2L, T), ready to send), ((W1-T2L, L), atm), ((W1-L2R, L), vac), ((W1-L2R, R), @l), ((W1-R2A, R), @a), ((W1-R2A, A), idle), ((W1-AL, A), idle), ...}.
- Se_2 = {((W1-T2L, T), ready to receive), ((W1-T2L, L), atm), ...}.

Note that by convention, state names are shown in lower case whereas task names are in upper case and start with the associated material instance id.

Although the generalized job shop scheduling definition described above forms a good basis for the scheduling problem in a complex machine, some essential constraints are missing to avoid infeasible schedules. Some of them have to do with material logistics. For instance, whereas material transport is feasible from the track to any of the lock resources of the lock capability, this is not the case from any lock to any robot: *logistic flow feasibility*. Moreover if, for instance, a wafer is transported into one of the locks it must be assured that it is taken from the same lock: *logistic flow integrity*. Furthermore, it must be assured that not too many wafers are in one of the locks at the same time as physical room does not allow for that: *material capacity feasibility*.

Also resource interference causes additional constraints. Unlike in job shop scheduling, a resource state transition may be constrained to be executed synchronously with other resource state transitions only, for instance the stage swap. On the other hand, multiple state transitions may be constrained to be executed one at a time as they involve visiting the same hazardous area, for instance robot rotations in front of the locks. Both these complications may be appropriate for part of a transition only, for instance the robot rotation from state @a to state @l visits the hazardous area between the intermediate state @ca and state @l only. Therefore, a possibly compound state transition must be decomposed into elementary state transitions. The same holds for the state transition of a wafer stage from @e to @lm, which must go via state @u.

Finally, the required nanometer accuracy imposes constraints in the form of time windows. As the wafer is conditioned on the alignment unit and the stage (but not between them), the time interval should not be no longer than necessary. This means that tasks A2R and R2S should preferably be executed without delay. Furthermore, the time between exposure and transport to the track (Post Exposure Bake time) should be as constant as possible, to achieve good imaging uniformity. Other examples of time windows can be found in [5].

3.3 Selecting resource assignment and task order

In the first subsection below, additional machine-specific constraints are introduced to avoid infeasible selections. In the second subsection, the constraints to be taken into account in a constructive selecting algorithm are described.

3.3.1 Machine-specific scheduling constraints

In the previous section, three machine-specific logistic integrity constraints involving material were introduced: logistic flow feasibility, logistic flow integrity and material capacity feasibility. An attempt to ensure material capacity feasibility for a job shop scheduling problem could be to instantiate resources for each storage location. Furthermore, storage location occupation tasks could be introduced that start as soon as a material is transported onto the resource associated with the storage location, and end when the material is transported off it again. However, as choices with respect to resource assignment also play a role in complex manufacturing machines, this attempt does not succeed in this case. The question remains how to make sure that the logistic flow is feasible, such that material can be transported from L0 to R0 but not to R1, or how to preserve logistic

flow integrity such that if material is transported from T0 into L0, it is transported from the same L0 and not from L1 to R0 afterwards.

The notion of material is added to TRS definition level 2 to describe the logistic integrity restrictions in an intuitive way.

The following five elements are added to the unselected TRS definition $\mathcal{D}_2$:

- $\mathcal{M}$ is a finite set whose elements are called material instances.
- Cb_2, Ce_2: $TC \rightarrow \mathcal{P}(\mathcal{M})$ give the begin and the end material configuration of each capability involved in a certain task, where $TC = \{(t, c) | t \in \mathcal{T}_2, c \in I_2(t)\}$
- Rm: $\mathcal{R} \rightarrow \mathbb{N}$ gives the number of material instances that can reside on a certain resource.
- $Mf \subseteq \mathcal{R} \rightarrow \mathcal{R}$ represents the physically possible material flow as a set of tuples defining from which resource to which resource material can flow.

Additional constraints that have to be satisfied concerning $\mathcal{D}_2$:

C-2b It is assumed that the subsets of material instances involved in a task remain the same from the begin to the end of a task:
$(\forall t : t \in \mathcal{T}_2 : \{Cb_2(t, c) | c \in I_2(t)\} = \{Ce_2(t, c) | c \in I_2(t)\})$

This constraint implies that only closed systems are considered, which means that material does not enter or leave the system.

Let P_{2m}: $\mathcal{D}_2 \times \mathcal{M} \rightarrow \mathcal{P}(\mathcal{T}_2 \times \mathcal{T}_2)$ be a function describing for each material $m \in \mathcal{M}$ in a TRS definition $D_2 \in \mathcal{D}_2$, a precedence relation between related tasks (the material 'life') without redundant edges and with matching capabilities:

$$\begin{aligned} P_{2m}(D_2, m) = \; & \{(t, t') \\ & | \; (t, t') \in P_2 \\ & \wedge \{c | c \in I_2(t), m \in Ce_2(t, c)\} = \{c | c \in I_2(t), m \in Cb_2(t', c)\} \\ & \wedge \neg redundant(t, t', P_2) \\ & \} \end{aligned} \tag{3.1}$$

Above, function *redundant*: $\mathcal{T}_2 \times \mathcal{T}_2 \times \mathcal{P}(\mathcal{T}_2 \times \mathcal{T}_2) \rightarrow \mathbb{B}$ determines whether a precedence edge (t, t') is redundant in a precedence relation P:

$$redundant(t, t', P) = (\exists t'' : t'' \in \mathcal{T}_2, t'' \neq t, t'' \neq t' : path(t, t'', P) \wedge path(t'', t', P)) \tag{3.2}$$

Here, function *path*: $\mathcal{T}_2 \times \mathcal{T}_2 \times \mathcal{P}(\mathcal{T}_2 \times \mathcal{T}_2) \rightarrow \mathbb{B}$ determines whether there is a path between two tasks t and t' in a precedence relation P:

$$path(t, t', P) = \begin{cases} true & \text{if } t = t' \\ (\exists t'' : (t, t'') \in P : path(t'', t', P)) & \text{if } t \neq t' \end{cases} \tag{3.3}$$

The additional constraints that have to be satisfied to ensure logistic integrity for transformation B from $\mathcal{D}_2$ into $\mathcal{D}_1$ are as follows.

C-Bc *Logistic flow integrity*: the resources involved in life of material instance $m \in \mathcal{M}$ in a TRS definition $D_2 \in \mathcal{D}_2$ are matching:

$$\begin{array}{ll}(\forall t, t' & : (t,t') \in P_{2m}(D_2, m) \\ & : I_1(t) \cap \{r | r \in A(c), c \in I_2(t), m \in Ce_2(t,c)\} \\ & = I_1(t') \cap \{r | r \in A(c), c \in I_2(t), m \in Cb_2(t',c)\} \\ )\end{array} \tag{3.4}$$

C-Bd *Logistic flow feasibility*: the combination of involved resources in material transport is physically possible:

$$\begin{array}{ll}(\forall t, m, r_b, r_e & : t \in \mathcal{T}_1, m \in \mathcal{M}, r_b, r_e \in \mathcal{R} \\ & , \{r_b\} = I_1(t) \cap \{r | r \in A(c), c \in I_2(t), m \in Cb_2(t,c)\} \\ & , \{r_e\} = I_1(t) \cap \{r | r \in A(c), c \in I_2(t), m \in Ce_2(t,c)\} \\ & : r_b = r_e \vee (r_b, r_e) \in Mf \\ )\end{array} \tag{3.5}$$

C-Be *Material capacity feasibility*: the material capacity of a resource is not exceeded. To define this constraint, some additional functions must be introduced. Let $P_{1r} : \mathcal{D}_1 \times \mathcal{R} \rightarrow \mathcal{P}(\mathcal{T}_1 \times \mathcal{T}_1)$ be a function describing for each resource r in a TRS definition $D_1 \in \mathcal{D}_1$ a linear precedence relation between related tasks, where linear means that the related tasks form a chain:

$$\begin{array}{ll}P_{1r}(D_1, r) = & \{(t,t') \\ & | \ (t,t') \in P_1, I_1(t) \cap I_1(t') \neq \emptyset, \neg redundant(t,t',P_1) \\ & \}\end{array} \tag{3.6}$$

Let *tchainr*: $\mathcal{P}(\mathcal{T}_1 \times \mathcal{T}_1) \rightarrow \mathcal{T}_1^*$ be a function that returns the task chain corresponding with a linear precedence relation P_l.

$$tchainr(P_l) = \begin{cases} \varepsilon & \text{if } P_l = \emptyset \\ [t] +\!\!+ \ tchainr(P_l \backslash firstp(P_l)) & \text{if } P_l \neq \emptyset \wedge \{(t,t')\} = firstp(P_l) \end{cases} \tag{3.7}$$

Above, $a +\!\!+ b$ denotes concatenation of sequence a and b, and function *firstp*: $\mathcal{P}(\mathcal{T}_1 \times \mathcal{T}_1) \rightarrow \mathcal{P}(\mathcal{T}_1 \times \mathcal{T}_1)$ determines the first precedence edge in a linear precedence relation P_l:

$$firstp(P_l) = \{(t,t') | (t,t') \in P_l, (\nexists t'' : t'' \in \mathcal{T}_1 : (t'',t) \in P_l)\} \tag{3.8}$$

Let $S_m(r,s)$ be the material configuration of resource r after execution of task sequence s. Before executing any task, the material configuration of a resource is given and is defined as the initial material configuration: $S_m(r,\varepsilon) = S_{m-i}(r)$. The material configuration after execution of task sequence $s +\!\!+ [t]$ or st is defined as follows:

$$S_m(r,st) = S_m(r,s) \backslash Cb_2(t,c) \cup Ce_2(t,c), \tag{3.9}$$

where $r \in A(c)$.

Then the material capacity constraint for a TRS definition $D_1 \in \mathcal{D}_1$ can be defined as follows:

$$(\forall r, st : r \in \mathcal{R}, t \in \mathcal{T}_1, st \preceq tchainr(P_{1r}(D_1, r)) : |S_m(r,st)| \leq Rm(r)) \tag{3.10}$$

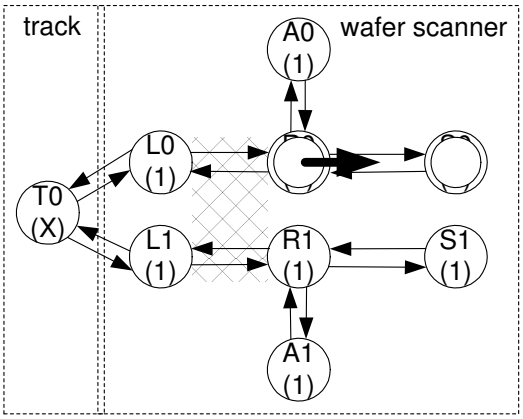


Figure 3.6: Deadlock situation in the example case

3.3.2 A constructive selecting algorithm

To avoid invalid behavior during constructive schedule generation, additional constraints are introduced that outline the valid extension of a selection.

WIP ceiling

To avoid deadlock, such as the situation displayed in Fig. 3.6, it is required to make sure that the number of material instances residing on a subset of the resources Rc does not exceed some number r_c.

C-Bf *Work In Progress ceiling.* When considering a constructive scheduling algorithm, a partial selection $D_{1p} \in \mathcal{D}_1$ can only be extended with a task t' and related definition elements to form an extended partial selection $D'_{1p} \in \mathcal{D}_1$ if no WIP ceiling constraints are violated for the extended partial selection.

Let $WIPceil \subseteq \mathcal{P}(\mathcal{R}) \times \mathbb{N}$ be the set of applicable combinations of Rc and n_c as described earlier. Then the additional constraint is defined as follows:

$$(\forall Rc, n_c : (Rc, n_c) \in WIPceil : (\Sigma r : r \in Rc : |S_m(r, tchainr(P_{1r}(D'_{1p}, r)))|) \leq n_c) \tag{3.11}$$

Tied precedences

In many cases, material transport is performed by resources that can contain only one material instance (one-lane logistic path). This means that the only possible next transport task for such a resource is to transport the material instance further. However, when a constructive scheduling algorithm is applied this can lead to deadlock. An example of this is illustrated in Fig. 3.7. To avoid such deadlock situations, the scheduling algorithm has to look some tasks further in life of this material instance than the next task only.

To describe these situations, the concept of tied precedences $Pt_2 \subseteq P_2$ is introduced. This implies that the subsequent transport tasks of a certain material instance that have to be executed without interrupting one another are connected by tied precedences. A *tie* is defined as a chain of tasks that are connected by tied precedences. An *open tie* is defined as a tie of which at least one - but not all - tasks are selected.

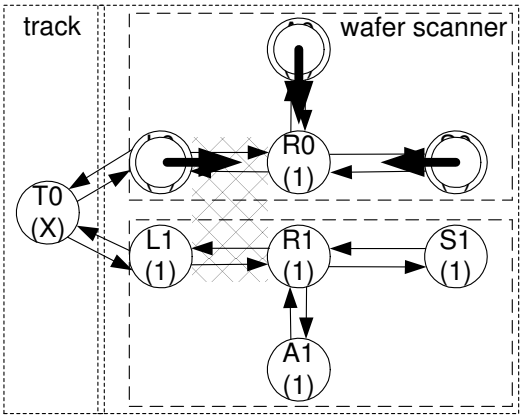


Figure 3.7: Another deadlock example in the example case

When considering a constructive scheduling algorithm, the additional constraint concerning ties is as follows:

C-Bg *Tied precedences.* A partial selection $D_{1p} \in \mathcal{D}_1$ can only be extended with a task t' from a tie to form an extended partial selection $D'_{1p} \in \mathcal{D}_1$ if it is possible to subsequently select an entire tie. If a partial selection contains an open tie, it can only be extended with a tied task.

Below, the possible schedule extensions for a constructive scheduling algorithm are formulated. Let $T_{1p},\ I_{1p}, P_{1p}$ be the tasks, involved resources and precedence relation of partial selection D_{1p}. Let $T_{1e},\ I_{1e}, P_{1e}$ be the task t', involved resources with t' and precedence edges to t' of selection extension D_{1e}. Let $T'_{1p},\ I'_{1p}, P'_{1p}$ be the tasks, involved resources and precedence relation of extended partial selection D'_{1p}, which is equal to $T_{1p} \cup T_{1e}, I_{1p} \cup I_{1e}, P_{1p} \cup P_{1e}$.

Let function $Et : (\mathcal{D}_2 \times \mathcal{D}_1) \rightarrow \mathcal{P}(\mathcal{T}_1)$ be a function that determines for a partial selection D_{1p} all eligible next tasks, considering the precedence relation.

$$Et(D_2, D_{1p}) = \{t \in \mathcal{T}_2 \backslash T_{1p} | (\forall t' : (t', t) \in P_2 : t' \in T_{1p})\} \tag{3.12}$$

Let function $Et_t : (\mathcal{D}_2 \times \mathcal{D}_1) \rightarrow \mathcal{P}(\mathcal{T}_1)$ be a function that determines for a partial selection D_{1p} all eligible tied next tasks, considering the tied precedence relation.

$$Et_t(D_2, D_{1p}) = \{t' | t' \in Et(D_2, D_{1p}), (\exists t : t \in T_{1p} : (t, t') \in P_{2t})\} \tag{3.13}$$

Let function $Er : (\mathcal{D}_2 \times \mathcal{T}_1) \rightarrow \mathcal{P}(\mathcal{R})$ be a function that determines for an eligible task t all eligible sets of involved resources, considering the available resources for the capabilities involved.

$$Er(D_2, t) = \{rr \subseteq \mathcal{R} \ | (\forall r, c : r \in rr, c \in I_2(t) : r \in A(c))\} \tag{3.14}$$

Let functions $check_{C-Ba}$ through $check_{C-Bc} : \mathcal{D}_2 \times \mathcal{D}_1 \times \mathcal{T}_1 \times \mathcal{P}(\mathcal{R}) \rightarrow \mathbb{B}$ be functions that check whether or not for a next task t' and involved resources rr' constraints C-Ba through C-Bc are satisfied.

$$\begin{aligned} check_{C-Ba}(D_2, D_{1p}, t', rr') = \ & (\forall t \ : \ t \in T_{1p} \wedge (t, t') \in P_{2m}(m) \\ & \quad : \ I_1(t) \cap \{r | r \in A(c), c \in I_2(t), m \in Ce_2(t, c)\} \\ & \quad \ \ = rr' \cap \{r | r \in A(c), c \in I_2(t), m \in Cb_2(t', c)\} \\ &) \end{aligned} \tag{3.15}$$

$$
\begin{aligned}
&check_{C-Bb}(D_2, D_{1p}, t', rr') = \\
&\quad (\forall m, r_b, r_e \;:\; m \in \mathcal{M}_2 \wedge r_b, r_e \in \mathcal{R}_1 \\
&\qquad\quad , \; \{r_b\} = rr' \cap \{r | r \in A(c), c \in I_2(t), m \in Cb_2((t,c))\} \\
&\qquad\quad , \; \{r_e\} = rr' \cap \{r | r \in A(c), c \in I_2(t), m \in Ce_2((t,c))\} \\
&\qquad\quad : \; r_b = r_e \vee (r_b, r_e) \in Mf \\
&\quad)
\end{aligned}
\tag{3.16}
$$

$$
check_{C-Bc}(D_2, D_{1p}, t', rr') = (\forall r : r \in rr' : |S_m(r, tchainr(P1r(D_{1p}, r)) +\!\!+ [t'])| \leq Rm(r))
\tag{3.17}
$$

Let function $check_{C-Bd} : \mathcal{D}_2 \times \mathcal{D}_1 \times \mathcal{T}_1 \times \mathcal{P}(\mathcal{R}) \times \mathcal{P}(\mathcal{P}(\mathcal{R}) \times \mathbb{N}) \rightarrow \mathbb{B}$ be a function that checks whether or not for a next task t' and involved resources rr' constraint C-Bd is satisfied.

$$
\begin{aligned}
&check_{C-Bd}(D_2, D_{1p}, t', rr', WIPceil) = \\
&(\forall Rc, n_c : (Rc, n_c) \in WIPceil : (\Sigma r : r' \in Rc \cap rr' : |S_m(r, tchainr(D_{1p}, r) +\!\!+ [t'])|) \leq n_c)
\end{aligned}
\tag{3.18}
$$

Let $E : (\mathcal{D}_2 \times \mathcal{D}_1) \rightarrow \mathcal{P}(\mathcal{T}_1 \times \mathcal{P}(\mathcal{R}) \times \mathcal{P}(\mathcal{T}_1 \times \mathcal{T}_1))$ be the function that, given an unselected TRS definition, returns all possible extensions e with which partial schedule D_{1p} can be extended to form an extended partial schedule D'_{1p}. Such an extension e is in the form of task t', involved resources rr' and precedences pr'. The extension definition D_{1e} can be determined from e by taking the first element from it, $e.0$, for T_{1e}, the first and second element, $(e.0, e.1)$, for I_{1e}, and the third element, $e.2$, for P_{1e}. Then, function E can be defined as follows:

$$
\begin{aligned}
&E(D_2, D_{1p}) = \\
&\{(t', rr', V) \\
&| V = \; \{(t,t') | (t,t') \in P_2 \\
&\qquad\qquad \vee ((I_{1p}(t) \cap rr' \neq \emptyset) \wedge (\nexists t'' \in T_{1t} : I_{1p}(t) \cap I_{1p}(t'') \neq \emptyset \wedge (t, t'') \notin P_{1p})) \\
&\qquad \} \\
&, (Et_t(D_{1p}) = \emptyset \Rightarrow t' \in Et(D_2, D_{1p})) \wedge (Et_t(D_{1p}) \neq \emptyset \Rightarrow t' \in Et_t(D_2, D_{1p})) \\
&, rr' \in Er(D_2, t') \\
&, check_{C-Ba}(D_2, D_{1p}, t', rr') \\
&, check_{C-Bb}(D_2, D_{1p}, t', rr') \\
&, check_{C-Bc}(D_2, D_{1p}, t', rr') \\
&, check_{C-Bd}(D_2, D_{1p}, WIPceil, t', rr') \\
&, (\exists D'_{1e} : D'_{1e} \in \mathcal{D}_1 : Et_t(D'_{1p} \cup D'_{1e}) = \emptyset) \\
&\}
\end{aligned}
\tag{3.19}
$$

In function E, both the constraints involved in generalized job shop scheduling, and the additional machine-specific scheduling constraints C-Ba through C-Bg can be recognized.

3.4 Timing the selected tasks

In the first subsection below, additional machine-specific constraints are introduced to avoid infeasible timing behavior. In the second subsection, the transformation function is described.

3.4.1 Machine-specific timing constraints

To avoid resources interfering with one another, some additional definition elements are introduced. Subsequently, constraints are defined using these additional elements. Furthermore, the constraint that must be satisfied to be able to time a selected TRS is described. Finally, constraints with respect to task start and finish times are described.

Additional TRS definition elements

Forced synchronism

Some state transitions of some resources can only take place synchronously with state transitions of other resources.

- $Ts \subseteq \mathcal{P}(\mathcal{R} \times \mathcal{S} \times \mathcal{S})$ gives the subsets of synchronous resource state transitions.

Collision avoidance

In a machine, certain areas exist in which resources can collide. These areas should be visited by the resources only one at a time. This additional constraint can be described by adding a resource to $\mathcal{R}$ for such a hazardous area, and involve this resource in every resource state transition that visits this area as described in the previous subsection. For the collision area resources, physical states do not play a role.

- $\mathcal{R}_c$ is a finite set whose elements are called collision areas.
- $Tc : \mathcal{R} \times \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{R}_c \cup \{\emptyset\}$ gives the collision area resource that is associated with a certain resource state transition.

Compound state transitions

The state transitions might consist of several elementary sub transitions, each of which might be involved with forced synchronism or collision avoidance.

- $Te : \mathcal{R} \times \mathcal{S} \times \mathcal{S} \rightarrow (\mathcal{R} \times \mathcal{S} \times \mathcal{S})^*$ gives the sequence of elementary sub transitions of a resource state transition. If there are no elementary sub transitions, the original state transition is returned.

Additional TRS definition constraints

C-a Every task is elementary:

$$(\forall t, r\colon t \in \mathcal{T}_1, r \in I_1(t) \cap (\mathcal{R} \backslash \mathcal{R}_c) \\ : Te(r, Sb_1(t,r), Se_1(t,r)) = [(r, Sb_1(t,r), Se_1(t,r))])$$

C-b Every task matches Ts, which implies that for each task t either the resource state transitions involved encapsulate some set of synchronous state transitions s from Ts, or none of the involved resource state transitions occurs in any s from Ts.

$$(\forall t : t \in \mathcal{T}_1 \quad : (\exists s \in Ts : s \subseteq (\cup r : r \in I_1(t) : \{(r, Sb_1(t,r), Se_1(t,r))\})) \\ \vee (\forall s : s \in Ts : s \cap (\cup r : r \in I_1(t) : \{(r, Sb_1(t,r), Se_1(t,r))\}) = \emptyset))$$

C-c Every task matches Tc, which implies that for each task and each involved resource (excluding hazardous areas) goes that for each resource state transition any involved collision area is involved in the task too:

$$(\forall t, r : t \in \mathcal{T}_1 \wedge r \in I_1(t) \cap (\mathcal{R} \backslash \mathcal{R}_c) : Tc(r, Sb_1(t, r), Se_1(t, r)) \in I_1(t))$$

Note that these constraints essentially hold for every TRS definition level (main = level 1).

C-Ae A selected TRS $D_1 \in \mathcal{D}_1$ is *timeable* if subsequent task end and begin states in the chain of tasks per resource match:

$$(\forall t, t', r\text{: } (t, t') \in tchainr(P_{1r}(D_1, r)), r \in I_1(t) \cap I_1(t') \cap (\mathcal{R} \backslash \mathcal{R}_c) \\ : Se_1((t, r)) = Sb_1((t', r)))$$

Note that although the FSM in Fig. 3.3 allow a number of possible sequences of elementary resource state transitions between some possible resource state transition that is implied by selection, only one is defined by function Te. Furthermore, for each elementary state transition at most one collision area is defined by function Te. As this constraint does not involve any selection and is a prerequisite for the timing transformation, it is categorized as a constraint on the timing transformation.

When taking the issue of forced synchronous and elementary state transitions into account, it is possible that a state transition of a resource implies state transitions of other resources. These implied state transitions also have to match the states of the resource in turn, which might imply other state transitions, and so on. To avoid an infinite chain reaction caused by loops, additional constraints are defined.

A *core state transition* is defined as the resource state transition of a resource r from the end state of the previous task on r to the begin state of the next task on r, in case these states do not match. Using this definition, the constraints described below have to be satisfied to prevent loops during the transformation into a timeable TRS.

- For two core state transitions necessary for one task t, the sets of resources involved in state transitions implied by each core state transition do not overlap. For example, when there are two core state transitions for resource A and B, and the core state transition for resource A implies a synchronous state transition of resource C, then the state transitions implied by the core state transition for resource B may not involve resource C.
- For any state transition of resource r' (either core or implied by other state transitions), the set of resources involved in state transitions implied by this state transition does not contain r' itself. For example, it is not allowed that a state transition of resource B that is implied by a state transition of resource A on its turn implies a state transition of resource A.

To conclude, a final machine-specific timing constraint is defined:

C-Af To transform a timeable selected TRS D_1 to a timed TRS D_0, besides the constraints presented in Section II, additional time constraints can be introduced for the time between certain task start or end times. Examples of these time windows from the example case are the Post Expose Bake time and the time that a wafer resides at a load robot.

3.4.2 Timing algorithm

Due to the constraints that ensure one possible finite transformation of a selected TRS D_1 to a timeable selected TRS $D_1^{\intercal}$, this transformation can be defined by a function. In the chain of tasks per resource of a definition D_1, additional tasks are introduced such that a chain of tasks results that satisfies constraints C-a through C-c and C-Ae to result in a timeable selected TRS $D_1^{\intercal}$.

Let *insert* be a function inserting tasks and precedence edges in a selected TRS D_1 for all non-matching subsequent task end and begin states defined by
$insert(D_1) = D_1'$ such that:

$$
\begin{array}{l}
(\forall r, t, t' \colon r \in \mathcal{R}, t, t' \in \mathcal{T}_1, (t, t') \in P_{1r}(D_1, r), Se_1(r, t) \neq Sb_1(r, t') \\
\qquad : (t, t') \notin P_1' \\
\qquad \wedge (\exists t'' \colon t'' \in \mathcal{T}_1', (t, t'') \in P_1', (t'', t') \in P_1' \\
\qquad\qquad : I_1'(t'') = \{r\} \wedge Sb_1'(t'', r) = Se_1(t, r) \wedge Se_1'(t'', r) = Sb_1(t', r) \\
\qquad) \\
)
\end{array}
\tag{3.20}
$$

and $\mathcal{T}_1'$, P_1', I_1', Sb_1', and Se_1' are minimal.

Let $et = Te(r, Sb_1(t, r), Se_1(t, r))$. Let *decomp* be a function decomposing any compound transitions in tasks of a selected TRS to match Te defined by
$decomp(D_1) = D_1'$ such that:

$$
\begin{array}{l}
(\forall t, r \colon t \in \mathcal{T}_1, r \in I_1(t), Te(r, Sb_1(r, t), Se_1(r, t)) \neq (r, Sb_1(r, t), Se_1(r, t)) \\
\qquad :(\forall 0 \leq i < len(et) \\
\qquad :(\exists t' \colon t' \in \mathcal{T}_1' \\
\qquad\qquad : I_1'(t') = \{r\} \wedge Sb_1'(t', r) = e.i.1 \wedge Se_1'(t', r) = e.i.2 \\
\qquad\qquad \wedge (i = 0 \Rightarrow (\forall t'' : t'' \in \mathcal{T}_1, (t'', t) \in P_{1r}(D_1, r) : (t'', t) \notin P_1' \wedge (t'', t') \in P_1')) \\
\qquad\qquad \wedge (i = len(et) - 1 \Rightarrow \ (\forall t'' \colon t'' \in \mathcal{T}_1, (t, t'') \in P_{1r}(D_1, r) \\
\qquad\qquad\qquad\qquad\qquad\qquad : (t, t'') \notin P_1' \wedge (t', t'') \in P_1' \\
\qquad\qquad\qquad\qquad\qquad) \\
\qquad\qquad) \\
\qquad) \\
\qquad) \\
)
\end{array}
\tag{3.21}
$$

and $\mathcal{T}_1'$, P_1', I_1', Sb_1', and Se_1' are minimal.

Let *notsync*: $\mathcal{D}_1 \rightarrow \mathcal{T}_1$ be a function determining which tasks of a selected TRS are not matching the forced synchronism element Ts:

$$
notsync(D_1) = \{t \in \mathcal{T}_1 \mid (\exists s : s \in Ts : s \nsubseteq \{(r, Sb_1(t, r), Se_1(t, r)) | r \in I_1(t)\})\} \tag{3.22}
$$

Let *addsync* be a function adding forced synchronous resource state transitions to tasks of a selected TRS which are not matching Ts defined by

$addsync(D_1) = D_1'$ such that:

$$\begin{array}{l}(\forall t \quad : t \in notsync(D_1) \\ \qquad : \quad (\forall s, ts \quad : s \in Ts, ts \in s \\ \qquad\qquad\qquad : ts \cap \{(r, Sb_1(t,r), Se_1(t,r)) | r \in I_1(t)\} \neq \emptyset \\ \qquad\qquad\qquad \Rightarrow ts.0 \in I_1'(t) \wedge ts = \{(r, Sb_1'(t,r), Se_1'(t,r))\} \\ \qquad\quad) \\ )\end{array} \tag{3.23}$$

Let *addcoll* be a function adding collision areas to tasks of a selected TRS that are not according Tc: $addcoll(D_1) = D_1'$ such that:

$$\begin{array}{l}(\forall t, r\text{: } t \in \mathcal{T}_1, r \in I_1(t) \cap (\mathcal{R} \backslash \mathcal{R}_c), Tc(r, Sb_1(t,r), Se_1(t,r)) \neq \emptyset \\ \qquad : Tc(r, Sb_1(t,r), Se_1(t,r)) \in I_1'(t) \\ )\end{array} \tag{3.24}$$

To transform a selected TRS D_1 that does not satisfy constraint C-Ae into a timeable selected TRS $D_1^{\intercal}$, function $trans_{A-t}$: $\mathcal{D}_1 \rightarrow \mathcal{D}_1$ is defined as follows:

$$trans_{A-t}(D_1) = \begin{cases} addcoll(decomp(insert(D_1))) & \text{if } notsync(decomp(insert(D_1))) = \emptyset \\ trans_{A-t}(addsync(decomp(insert(D_1)))) & \text{if } notsync(decomp(insert(D_1))) \neq \emptyset \end{cases} \tag{3.25}$$

The algorithm to optimize timing of a timeable selected TRS $D_1^{\intercal}$ given the timing constraints is a linear programming (LP) problem [8]. The variables of the LP problem are the start and finish times of each task, τ_{S_0} and τ_{F_0}, whereas the constraints are defined by C-Ab and C-Ac. Constraint C-Ad is not necessary anymore, as all state transitions in a timeable selected TRS are tasks.

3.5 Results

In this section, the theory presented in the previous sections is applied to the example case. For the example system, the additional elements can be instantiated as follows:

- $\mathcal{M}$ = {W1, W2, ...}.
- Cb_2 = {((W1-T2L, T), {W1}), ((W1-T2L, L), {}), ((W1-L2R, L), {W1}), ((W1-L2R, R), {}), ...}.
- Ce_2 = {((W1-T2L, T), {}), ((W1-T2L, L), {W1}), ...}.
- Rm = {(T0, 1), (L0, 1), ...}. This element can easily be extracted from Fig. 3.2.
- Mf = {(T0, L0), (L0, R0), (R0, A0), ...}. This element can easily be extracted from Fig. 3.2.
- Pt_2 = {(W1-T2L, W1-L2R), (W1-L2R, W1-R2A), (W1-R2A, W1-AL), (W1-AL, W1-A2R), (W1-A2R, W1-R2S), (W1-S2R, W1-R2L), (W1-R2L, W1-L2T), ...}. These precedence relations concern all transport tasks involving the robot.

- $WIPceil$ = {({L0, R0, A0, S0}, 2), ({L1, R1, A1, S1}, 2)}.
- Ts = {{(S0, @lm, @e), (S1, @e, @lm)}, {(S1, @lm, @e), (S0, @e, @lm)}}.
- Rc = {HA}.
- Tc = {((R0, @ca, @l), HA), ...}.
- Te = {((R0, @l, @a), [(R0, @l, @ca), (R0, @ca, @a)]), ((S0, @lm, @u), [(S0, @lm, @e), (R0, @e, @u)]), ...}. These sequences can be easily be extracted from Fig. 3.3.

Using a simple heuristic, such as 'fill up the system as much as possible and schedule tasks that can start earliest first', a feasible and valid schedule is obtained. Subsequently, a timing postprocessing step is done to fulfill the additional time windows: τ_{F_0}(W1-R2S) - τ_{S_0}(W1-A2R) = 3 [sec] etc., and τ_{F_0}(W1-L2T) - τ_{S_0}(W1-EXP) = 50 [sec] etc. After this, the schedule of Fig. 3.8 is obtained. It can be concluded that the schedule satisfies all additional restrictions. Note that these constraints can not be fulfilled without prediction.

The critical path shown in Fig. 3.9: exposure (EXP) is on the critical path in steady-state operation, which is as desired.

3.6 Conclusions

Generalized job shop scheduling can form a basis for scheduling in complex machines. However, to account for machine-specific restrictions such that only feasible schedules result, additional scheduling constraints are introduced. To this end, the job shop scheduling model is extended with some elements, based on which scheduling constraints are defined.

During the selecting phase of scheduling, the additional constraints ensure feasible behavior concerning material logistics. To enable partial dispatching, partial schedules are not allowed to lead to deadlock situations, as any backtracking is not possible after dispatching. However, as a consequence of the constraints introduced, deadlock is possible. Additional selecting constraints are introduced to avoid such invalid behavior. Furthermore, additional timing constraints are introduced to ensure feasible behavior concerning resource interference. Also the constraints concerning time windows are incorporated.

Defining manufacturing recipes using graphs is intuitive and offers great expressivity concerning recipes. The approach is flexible for strongly recipe dependent products and covers the entire machine (all tasks and resources, steady-state and transient behavior). Within the constraints introduced and the available scheduling time, real-time optimization of machine behavior is possible. In extreme time-critical situations, partial schedules consist of only one task: no prediction. In this case, the approach is similar to (current) state-based supervisory control [11, 13]. However, these approaches are not flexible for handling multiple product types at the same time. The approach presented in this chapter combines the benefits of both the optimization techniques from scheduling community and the behavior validity and responsiveness from (supervisory) control community.

The applicability of the instantiated, unselected TRS and the possible behavior improvement is illustrated using an example from a wafer scanner. Results show that instantiation of the machine-specific additional definition elements is straightforward. The additional elements involving selecting can be taken from a schematic layout of the machine, whereas the elements involving timing can be extracted from Finite State Machines (FSM) of the resources.

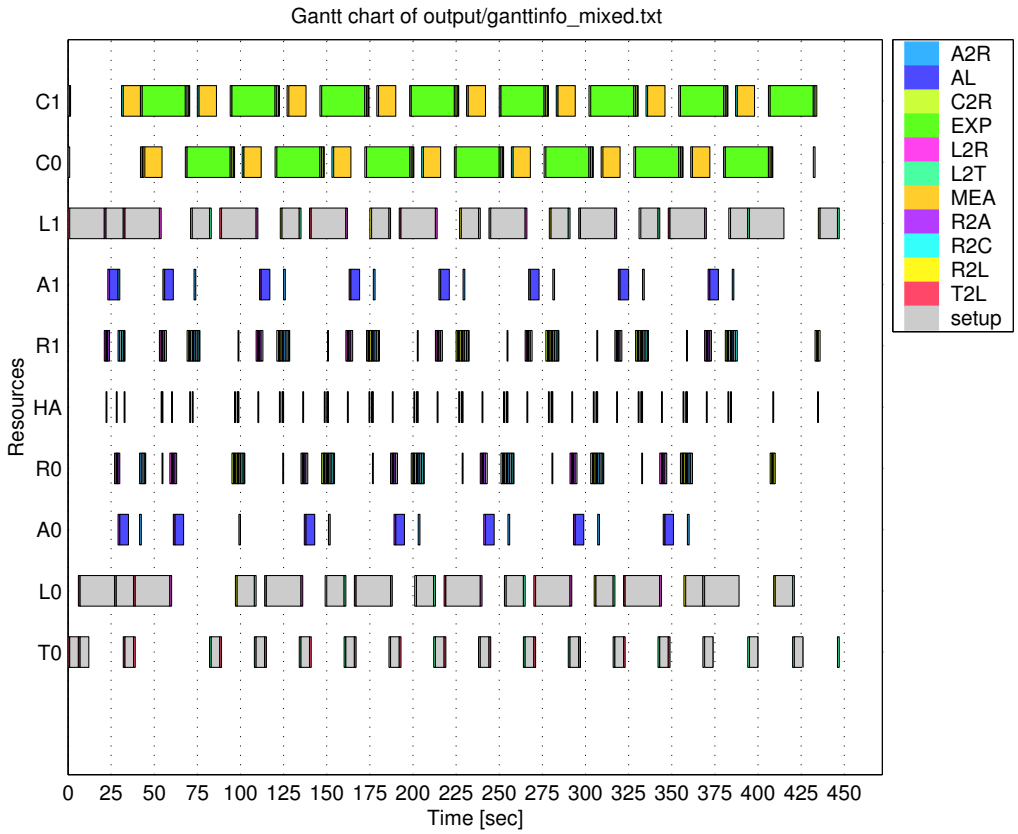


Figure 3.8: A resulting schedule for the example case

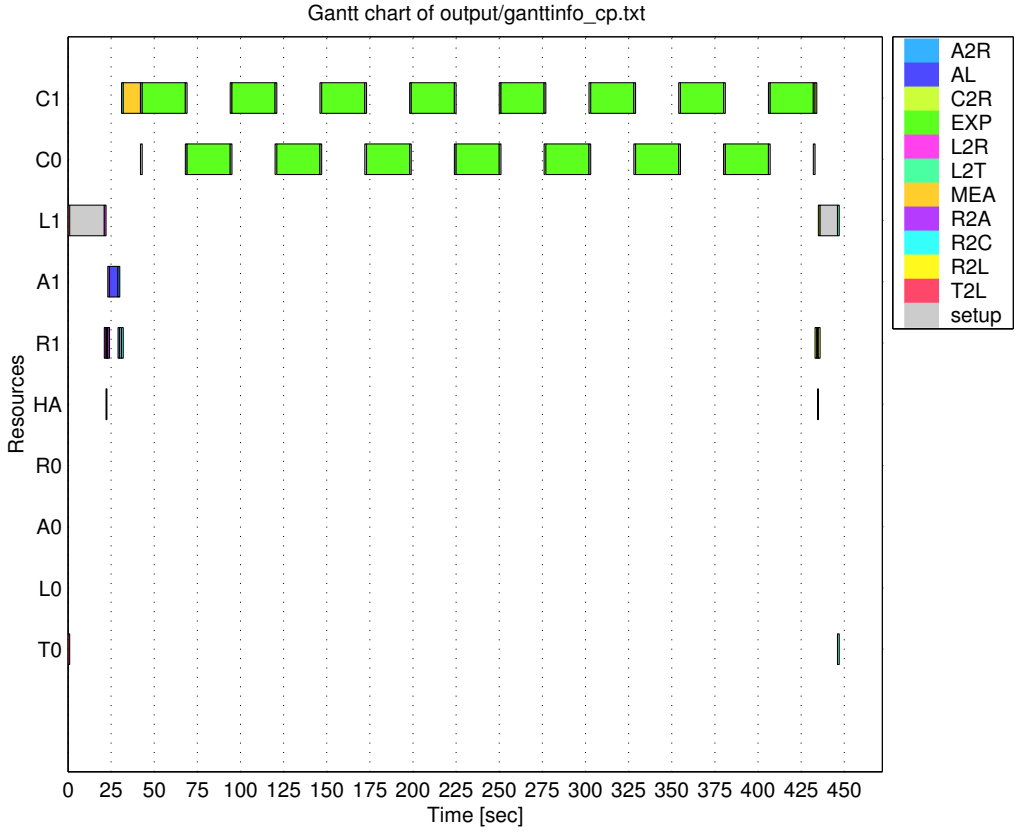


Figure 3.9: Critical path of a resulting schedule for the example case

The following open issues remain. To be absolutely sure that no deadlock schedules will be generated, verification of whether the definition elements involved are instantiated correctly is essential. Furthermore, the instantiation functionality (C) is to be developed to be able to react on triggers like manufacturing orders and exceptions by instantiating TRS definitions. These open issues are subject of current research.

Acknowledgments

The authors would like to acknowledge Cor Hurkens for his valuable comments.

References

[1] ASML, 2004. Information on wafer scanners available through URL `http://www.asml.com/`, item: products - lithography.

[2] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–341, 1994.

[3] P. Gohari and W. M. Wonham. Reduced supervisors for timed discrete-event systems. *IEEE Transactions on Automatic Control*, 48(7):1187–1198, 2003.

[4] D. Jevtic. Method and apparatus for automatically generating schedules for wafer processing within a multichamber semiconductor wafer processing tool, 1997. Patent no. US 6,201,999.

[5] J. Kim, T. Lee, H. Lee, and D. Park. Scheduling analysis of time-constrained dual-armed cluster tools. *IEEE Transactions on Semiconductor Manufacturing*, 16(3):521–534, 2002.

[6] S. Kumar, N. Ramanan, and C. Sriskandarajah. Robotic system control, 2003. Patent no. US 6,556,893.

[7] H. Marchand, O. Boivineau, and S. Lafortune. On the synthesis of optimal schedulers in discrete-event control problems with multiple goals. *SIAM Journal on Control Optimization*, 39(2):512–532, 2000.

[8] K. G. Murty. *Linear Programming*. Wiley-Interscience, Chichester, 1983.

[9] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, and J. E. Rooda. Design of supervisory machine control. In K. Glover and J. Maciejowski, editors, *Proceedings of the European Control Conference 2003*, 2003. CD-ROM.

[10] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 1995.

[11] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[12] S. Rostami and B. Hamidzadeh. Optimal scheduling techniques for cluster tools with process-module and transport-module residency contraints. *IEEE Transactions on Semiconductor Manufacturing*, 15(3):341–349, 2002.

[13] Y. Shin, T. Lee, J. Kim, and H. Lee. Modeling and implementing a real-time scheduler for dual-armed cluster tools. *Computers in Industry*, (45):13–27, 2001.

[14] S. F. Smith. Is scheduling a solved problem? In G. Kendall, E. Burke, and S. Petrovic, editors, *Multidisciplinary International Conference on Scheduling : Theory and Applications(MISTA'03)*, pages 11–20. ASAP, University of Nottingham, UK, August 2003.

[15] G. E. Vieira, J. W. Herrmann, and E. Lin. Rescheduling manufacturing systems: a framework of strategies, policies and methods. *Journal of scheduling*, 6(1):35–58, 2003.

[16] M. Wennink. *Algorithmic Support for Automated Planning Boards*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1995.

CHAPTER

FOUR

REACTION SCENARIOS INCLUDING EXCEPTION RECOVERY

This chapter contains the paper *Predictive-Reactive Scheduling in Complex Manufacturing Machines: Reaction Scenarios Including Exception Recovery* that has been protected in patent application ASML ref. P – 1885. First filing was in the US in May 2004, number 10/852,678.

Predictive-reactive scheduling in complex manufacturing machines: reaction scenarios including exception recovery

N.J.M. van den Nieuwelaar †*, J.M. van de Mortel-Fronczak †,
R. Boumen †, J.E. Rooda †

Abstract

Supervisory Machine Control (SMC) is responsible for a proper reaction to all kinds of triggers from its environment by deciding when to do which tasks using which resources. A predictive-reactive supervisory control concept is proposed, embedding existing scheduling functionality. In this concept, SMC consists of two parts: a predictive part and a dispatching part. The predicting part contains planning functionality to instantiate scheduling problems using a scheme of predefined construction rules. The scheme of planning rules describes how to transform triggers into scheduling problems step by step by adding predefined scheduling problem building blocks. After scheduling of the resulting problem, the schedule is dispatched to the resources by the dispatching part. Several scenarios are described to react to different control triggers, including exception recovery to react to tasks that fail. To avoid control overhead, prediction takes place in parallel with dispatching if possible. The control concept is illustrated using a representative example of a complex manufacturing machine: a wafer scanner.

4.1 Introduction

The purpose of a manufacturing machine is to make products, which requires physical manufacturing processes to be carried out. To actually do the work, mechatronic systems in the machine must be deployed. In complex manufacturing machines, many options exist to deploy the available resources to perform tasks that lead to the desired manufacturing purpose, resulting in various machine behaviors. Supervisory Machine Control (SMC) is responsible for deciding when to do which tasks using which resources. There are three important complicating requirements for SMC of complex manufacturing machines. First of all, the manufacturing tasks are heavily product recipe dependent, for which SMC must be flexible. It must be able to handle a stream of mixed product types, which are being processed concurrently. Second, SMC must be able to optimize machine behavior by exploiting its resources best within its manufacturing possibilities. What is best may depend on the characteristics of the recipe. Finally, it must fit in the dynamic environment that it is embedded in. This implies that it must react to all kinds of triggers from the environment without introducing unnecessary control overhead. A very important trigger is task failure: an exception, which requires a recovery reaction of SMC to avoid human intervention.

† Eindhoven University of Technology: P.O. box 513, 5600 MB Eindhoven, The Netherlands.
* ASML: De Run 6501, 5504 DR Veldhoven, The Netherlands.
Corresponding author: N.J.M. van den Nieuwelaar, e-mail: n.j.m.v.d.nieuwelaar@tue.nl

4.1.1 Literature

Many approaches exist to describe a system under supervisory control using well-known formalisms from computer science. Supervisory control theory (SCT) as discussed by Wonham et al. [3, 8, 17] models the system under control using Finite State Machines. The possible behavior of such a system is regarded as a language. A supervisory controller in the form of a deterministic automaton is synthesized that restricts the language by disabling a subset of events, to control the system to properly accomplish its task. Basic SCT theory does not meet the first requirement, as supervisors must be modelled specifically for the task to be accomplished, and therefore are not flexible for handling different recipes. Extensions to this theory have been developed to be able to control multiple predefined types of products with fixed routes [5, 18]. This still is not sufficient for our purpose as the various recipe parameters imply an infinite number of product types, which makes it impossible to predefine their route. Moreover, multiple routes might exist that lead the the desired manufacturing purpose, and fixed routes might exclude optimal behavior.

Optimization possibilities are restricted in SCT-based approaches, due to the lack of predictive information. Scheduling-based control concepts are better suited for this [4], especially a predictive-reactive scheduling approach [14]. In complex manufacturing machines multiple manufacturing possibilities may exist that all lead to the desired manufacturing purpose. This is in contrast with the predefined manufacturing sequences that are typically assumed, e.g. in [4, 14]. It is important that these scheduling possibilities can be expressed to be able to choose the optimal one. In Chapters 2 and 3 ([11, 12]), a predictive scheduling approach is presented that is suited for complex manufacturing machines. The scope of that work is a static scheduling problem. Any reaction to triggers from the environment is not addressed, which requires a predictive-reactive scheduling approach.

In the area of job shop scheduling, much research has been done on reactive scheduling [1, 6, 23]. An overview of rescheduling environments, strategies and methods can be found in [19, 22]. In that work, reaction on new work that arrives is straightforward: new jobs consisting of a fixed graph of processes are instantiated and added to the old scheduling problem. An important restriction of these rescheduling methods is that they do not cover job modification which is required for exception recovery. Exception recovery is discussed in SCT-based control approaches [7, 9, 16], but in this work only local recovery is considered: only one product is affected.

The remainder of this chapter shows that in the application area of complex manufacturing machines, the mapping of new work that arrives onto the scheduling problem is not straightforward. To add new work, it even can be interesting to modify the old scheduling problem to get a better overall schedule. Furthermore, non-local exception recovery is addressed, which also requires modification of the scheduling problem. The problem of determining which activities are to be done is called *planning* in literature [20]. In scheduling literature, planning typically is done in advance in a static setting. In the context of SMC, planning must be embedded in a dynamic environment. This chapter is based on Chapters 2 and 3 ([11, 12]) and adds the planning and dynamic reaction functionalities.

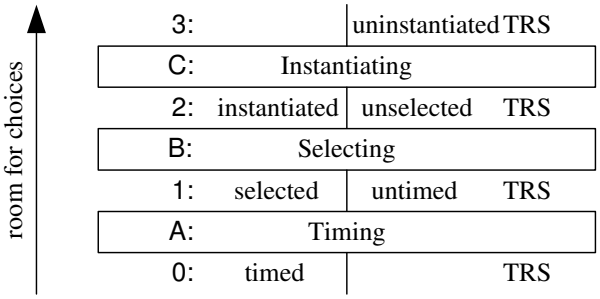


Figure 4.1: Layered Task Resource System framework

4.1.2 Layered task resource system framework

From the SMC point of view, a machine can be considered as a task resource system (TRS). Tasks can be associated with manufacturing processes, whereas resources can be associated with mechatronic systems. Transforming a manufacturing request into machine behavior can be structured in three phases. First, a scheduling problem must be instantiated from the manufacturing request, taking into account the limitations of the machine. This transformation is called *instantiating*. The structure of the resulting scheduling problem shows many similarities with the job shop scheduling problem [15]. The manufacturing process of a material instance can be associated with a job, whereas the different parallel mechatronic systems can be associated with the different machines in a job shop. Subsequently, resources must be assigned to the tasks in the instantiated scheduling problem in some order, taking into account the fact that resources are able to perform certain tasks only, and only one at a time. This transformation is called *selecting*. The selected order of tasks to be performed by selected resources implies consecutive state transitions of those resources, which is analogous to the setup times for mode switching in job shop scheduling. Finally, start and finish times can be assigned to the tasks, taking into account the speed of the resources. This transformation is called *timing*.

During the three transformation phases of instantiating, selecting and timing, choices must be made. The result of a choice in a certain transformation on the machine behavior can only be evaluated by performing the consecutive transformations. Therefore, a transformation phase strongly relies on information from subsequent phases. The layered TRS framework shown in Fig. 4.1 displays the hierarchically related transformation phases as functionality layers (A through C) and the different TRS definition levels (0 through 3) as interfaces between the layers (see Chapter 1 and [13]).

4.1.3 Structure of the chapter

The structure of this chapter is as follows. Throughout the chapter, an example of a complex machine is used for illustration: a dual-stage wafer scanner [2]. Section 4.2 describes the wafer scanner and instantiates a scheduling problem for a basic job. Section 4.3 describes the instantiation of a scheduling problem using a typical, more complex job as an example. Analysis shows that building blocks and construction rules can be distilled that describe how a user request is transformed into a scheduling problem (planning). The necessary functionality to glue the building blocks together is captured in functions, that are parameterized for the example for illustration. In Section 4.4, an SMC framework is

presented that embeds the layered TRS framework of Fig. 4.1. Different scenarios to react on several types of triggers are discussed: reaction to delays, to new work that arrives, but also to work that fails. Reaction to work that fails may require modification of the scheduling problem to support exception recovery. For each type of trigger, an example illustrates the result of the different reaction scenarios. Finally, concluding remarks are presented in Section 4.5.

4.2 Scheduling in a wafer scanner

4.2.1 A dual-stage wafer scanner

The primary manufacturing process of a wafer scanner is the exposure of a mask containing an IC pattern onto wafers. Fig. 4.2 shows a schematic layout of a dual-stage wafer scanner. In this figure, circles depict the mechatronic subsystems or resources regarded here, and arrows depict the possible transport paths. The number of material instances that a resource can carry is depicted between brackets. At the right side of the picture, the resources involved in the exposure process are shown. At the upper-right, the reticle stage (S0) that carries the mask containing the IC pattern is depicted. At the lower-right, the two wafer chucks that are present in a dual-stage wafer scanner (C0 and C1) are depicted, that carry the wafer during exposure. In between, a resource is depicted that stands for the required optics and the light source (O0).

Furthermore, the lower part of the figure deals with wafers, whereas the upper part deals with reticles. As the required accuracy of the exposure process is very high, any imperfections concerning the wafers must be corrected for. To be able to do this, wafers are measured at a wafer chuck before being exposed. The orientation of the wafer at a wafer stage is of importance for successful measurement and exposure, whereas the orientation is unknown when a wafer comes into the machine. Therefore, an alignment unit (A0) has been incorporated. A neighboring machine named track (T0) performs some pre-processing and post-processing steps, and delivers wafers to the alignment system. A load robot (L0) transports wafers from the alignment system to the wafer chucks. An unload robot (U0) transports wafers from the wafer chucks to the discharge unit (D0), from which wafers are picked up by the track.

Reticles enter and leave the system via the reticle pod (P0). Each reticle that has been in the pod must be inspected in the inspection station (I0) before it can be used for exposure. A buffer station (B0) is available that can be used to store inspected reticles. Transportation of reticles to and from the reticle stage is done by one of the two elevators (E0 and E1). Transportation of reticles between the pod, the elevators and the inspect and buffer stations is done by the reticle robot (R0).

4.2.2 Scheduling in a complex manufacturing machine

In Chapters 2 and 3 ([11, 12]), the scheduling model of a complex machine is defined. For the purpose of this chapter, the elements needed for the timing transformation (see Fig. 4.1) are not relevant. Without them, the scheduling model can be defined by an 18–tuple:

$(\mathcal{T}_2, \mathcal{L}_2, \mathcal{G}_2, \mathcal{N}_2, Ln_2, Gn_2, Ga_2, \mathcal{R}, \mathcal{C}, I_2, A, P_2, Pt_2, \mathcal{M}, Cb_2, Ce_2, Rm, Mf)$, where

- $\mathcal{T}_2$ is a finite set of elements called tasks.

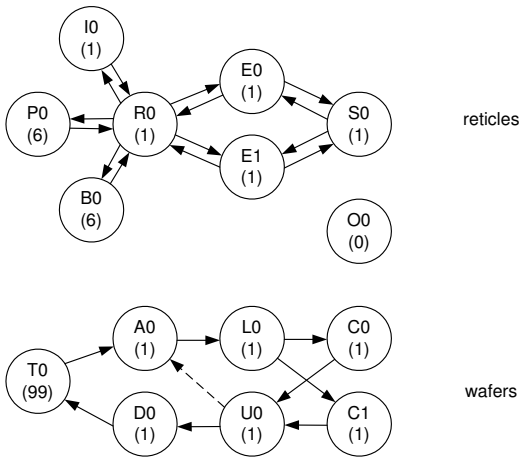


Figure 4.2: Schematic layout of a dual stage wafer scanner

- $\mathcal{L}_2$ is a finite set of elements called clusters.
- $\mathcal{G}_2$ is a finite set of elements called groups.
- $\mathcal{N}_2$ is a finite set of elements called nodes and is a generalization of the model elements mentioned earlier: $\mathcal{N}_2 = \mathcal{T}_2 \cup \mathcal{L}_2 \cup \mathcal{G}_2$.
- Ln_2: $\mathcal{L}_2 \rightarrow \mathcal{P}(\mathcal{N}_2)$ gives the set of nodes that are in a certain cluster.
- Gn_2: $\mathcal{G}_2 \rightarrow \mathcal{P}(\mathcal{N}_2)$ gives the set of nodes (alternatives) that a group consists of.
- Ga_2: $\mathcal{G}_2 \rightarrow \mathcal{P}(\mathbb{N})$ gives the allowed numbers (including 0) of nodes to be selected from a group.
- $\mathcal{R}$ is a finite set of elements called resources.
- $\mathcal{C}$ is a finite set of elements called capabilities.
- I_2: $\mathcal{T}_2 \rightarrow \mathcal{P}(\mathcal{C})$ gives the set of capabilities that are involved in a certain task.
- A: $\mathcal{C} \rightarrow \mathcal{P}(\mathcal{R})$ gives the set of resources that are available for a certain capability.
- $P_2 \subseteq \mathcal{N}_2 \times \mathcal{N}_2$ is the precedence relation between nodes.
- $Pt_2 \subseteq \mathcal{N}_2 \times \mathcal{N}_2$ is the tied precedence relation between nodes.
- $\mathcal{M}$ is a finite set whose elements are called material instances.
- Cb_2, Ce_2: $TC \rightarrow \mathcal{P}(\mathcal{M})$ give the begin and the end material configuration of each capability involved in a certain task, where $TC = \{(t, c) | t \in \mathcal{T}_2, c \in I_2(t)\}$
- Rm: $\mathcal{R} \rightarrow \mathbb{N}$ gives the number of material instances that can reside on a certain resource.
- $Mf \subseteq \mathcal{R} \times \mathcal{R}$ represents the physically possible material flow as a set of tuples defining from which resource to which resource material can flow.

Some of the elements have a suffix, which represents the definition level in Fig. 4.1. The elements that do not have a suffix are level independent.

4.2.3 An instantiated scheduling model

The scheduling model in a complex machine can be split into two sections: system-dependent elements and work-dependent elements. The system-dependent elements can be defined using Fig. 4.2 as follows:

- There are thirteen capabilities. For wafers: the chuck, the load and unload robots, the alignment and discharge units, and the track. For reticles: the stage, the elevator, the robot, the inspection and buffer station, and the pod. Besides that, there is a capability for the optics and light subsystems.
  $\mathcal{C}$ = {C, L, U, A, D, T, S, E, R, I, B, P, O}.
- There are fifteen resources: one for each capability, except two wafer chucks and two elevators:
  $\mathcal{R}$ = {C0, C1, L0, U0, A0, D0, T0, S0, E0, E1, I0, B0, R0, P0, O0}.
- The available resources for each capability are defined as follows:
  A = {(C, {C0, C1}), (L, {L0}), (U, {U0}), (A, {A0}), (D, {D0}), (T, {T0}), (S, {S0}), (E, {E0, E1}), (I, {I0}), (B, {B0}), (R, {R0}), (P, {P0}), (O, {O0})}.
- The material capacity of the resources is one for each resource, except for the track, the pod, and the buffer:
  Rm={(C0, 1), (C1,1), (L0,1), (U0,1), (A0,1), (D0,1), (T0,99), (S0,1), (E0,1), (E1,1), (I0,1), (B0,6), (R0,1), (P0,6), (O0,1)}.
- The possible material flow is defined as follows:
  Mf = {(T0, A0), (A0, L0), (L0, C0), (L0, C1), ...}.

To define the work-dependent elements, the steps in the manufacturing process of wafer *W1* and reticle *RA* are analyzed for one exposure. These 'basic' lives of a wafer and a reticle are depicted in the precedence graph of Fig. 4.3. First, the wafer is transported from the track onto the alignment unit (T2A). Subsequently, the alignment takes place (Ali). After that, the load robot takes the wafer from the alignment unit (A2L), and places the wafer onto a chuck (L2C). On the chuck, the wafer is measured (mea) and, subsequently, exposed (exp). Then, the unload robot takes the wafer from the chuck (C2U) and puts the wafer onto the discharge unit (U2D). Finally, the wafer is taken from the discharge unit by the track (D2T).

The life of a reticle consists of the following steps. The robot takes the reticle from the pod (P2R), and puts the reticle onto the inspection station (R2I), where it is inspected (Ins). After taking the reticle from the inspection station (I2R), there is a possibility to store the reticle (st?). Subsequently, the robot puts the reticle onto an elevator (R2E), that puts the reticle onto the reticle stage (E2S). At the stage, the reticle is used for exposure (exp). When exposure is done, an elevator picks the reticle up (S2E), after which the robot takes the reticle over (E2R) and puts it back into the pod (R2P). All nodes in Fig. 4.3 are tasks, except for the possible storage node (st?), that is depicted by a double circle. A drill-down of this node exists, that is shown at the right side of the figure. The node hierarchy is as follows. The upper node (st?) is a node of type group, of which zero or one alternatives must be chosen. The group node consists of one alternative of node type cluster (st!) consisting of two consecutive tasks, to describe transportation of the reticle to the buffer (R2B), followed by a transportation back (B2R).

For the basic material lives of wafer *W1* and reticle *RA* as depicted in Fig. 4.3, the work-dependent elements can be instantiated as follows:

- $\mathcal{T}_2$ = {W1-T2A, W1-Ali, ..., RA-P2R, RA-R2I, ...}.
- $\mathcal{L}_2$ = {RA-st!}.

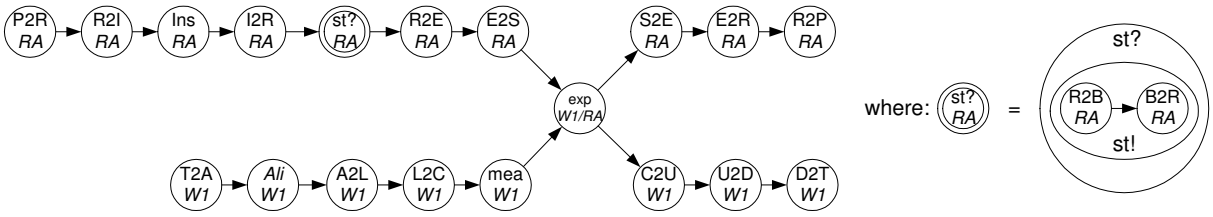


Figure 4.3: Basic lives of a reticle and a wafer

- $\mathcal{G}_2 = \{$RA-st?$\}$.
- $I_2 = \{$(W1-T2A, {T, A}), (W1-Ali, {A}), ... $\}$.
- $P_2 = \{$(W1-T2A, W1-Ali), (W1-Ali, W1-A2L), ... $\}$.
- $Pt_2 = \{$(RA-P2R, RA-R2I), (RA-I2R, RA-st?), (RA-st?, RA-R2E), (RA-E2R, RA-R2P)$\}$. These precedence relations concern all transportation tasks involving the reticle robot.
- $Ln_2 = \{$(RA-st!, {RA-R2B, RA-B2R})$\}$.
- $Gn_2 = \{$(RA-st?, {RA-st!})$\}$.
- $Ga_2 = \{$(RA-st?, {0, 1})$\}$.
- $\mathcal{M} = \{$RA, W1$\}$.
- $Cb_2 = \{$((W1-T2A, T), {W1}), ((W1-T2A, A), {}), ((W1-Ali, A), {W1}), ... $\}$.
- $Ce_2 = \{$((W1-T2A, T), {}), ((W1-T2A, A), {W1}), ... $\}$.

Note that by convention, task names are in upper case and start with the associated material instance id.

A schedule for this basic wafer scanner scheduling problem is shown in Fig. 4.4.

4.3 Planning

4.3.1 Planning a typical wafer scanner order

Under the pressure of the ever shrinking critical dimension in semiconductor industry, multiple exposure techniques are a trend in lithographic manufacturing. These techniques use multiple reticles to expose a single IC, each containing parts of the pattern. A typical example is the use of two reticles, one containing the coarse part of the circuit and one containing the small details. In this section, we analyse planning of such a dual exposure order for a batch of two wafers (order 1). The reticles involved are RA and RB, and exposure should be done according to the 'ABBA' pattern. This means that the first wafer is exposed using reticle RA first and then using RB, and for any next wafers, the reticle order used for exposure is alternating.

A logical first step in the planning process is to focus on the primary manufacturing process: exposure. For order 1, a sequence of four exposure steps can be determined: first expose wafer W1 with reticle RA, then expose wafer W1 with reticle RB, subsequently expose wafer W2 with reticle RB, and finally expose wafer W2 with reticle RA. Next, the secondary manufacturing processes can be added: the logistics and the pre-processing. Before a wafer can be exposed on a wafer chuck, a sequence of logistic input and pre-processing steps must be carried out on the wafer, as is explained in the previous section,

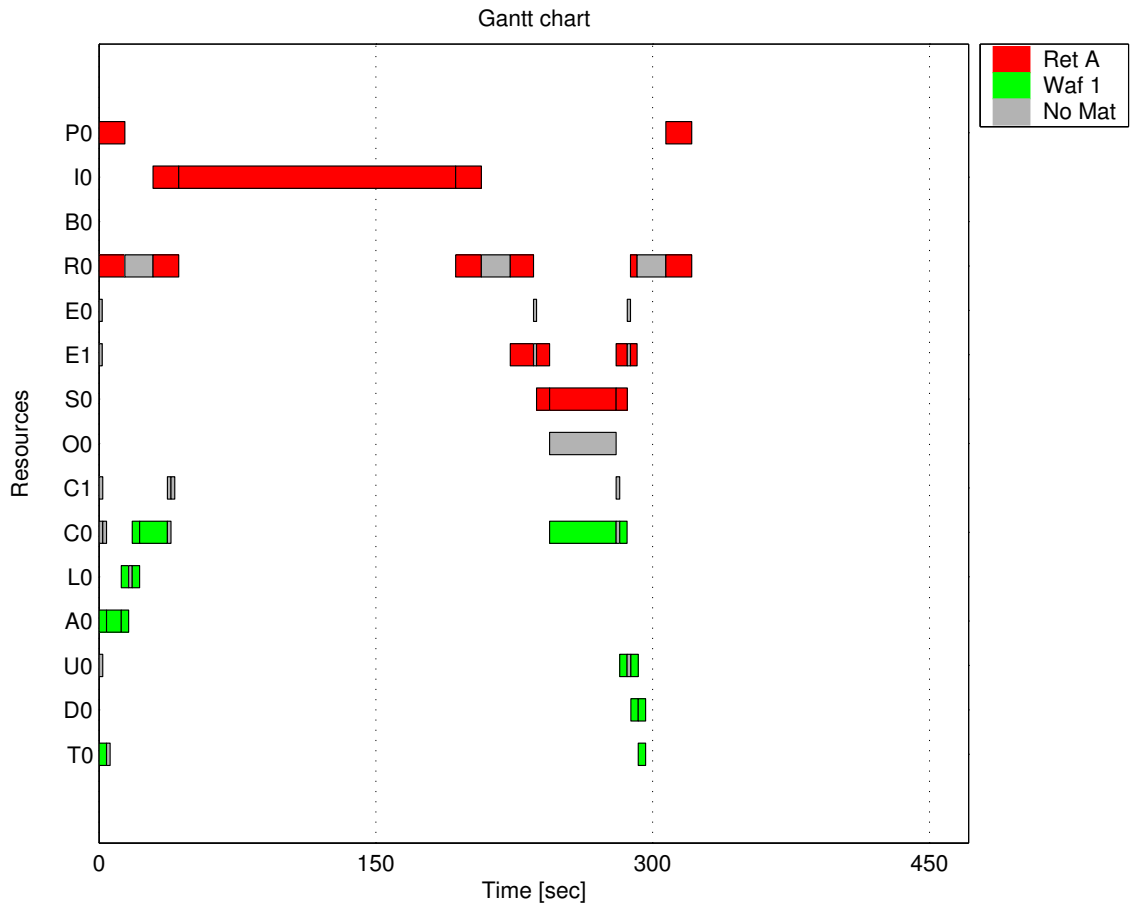


Figure 4.4: Basic schedule for one reticle and one wafer

and shown in the lower-left part of Fig. 4.3. This sequence of steps is called 'load wafer' from now. After exposure of a wafer, a sequence of logistic output steps must be carried out on the wafer, as is shown in the lower-right part of Fig. 4.3. This sequence is called 'unload wafer' from now. At the right side of Fig. 4.5, the precedence graph concerning wafers for order 1 is depicted. The four exposure steps are depicted in a dashed box. Before the first exposure step of each wafer, a 'load wafer' sequence is shown, whereas an 'unload' wafer sequence is shown after the latest exposure step of each wafer. As the track first delivers wafer W1 and then wafer W2, a precedence edge is drawn between the first two 'T2A' nodes.

At the left side of Fig. 4.5, the reticle view on the precedence graph for order 1 is depicted. For clarity, the dashed box containing the four exposure nodes is copied. Reticle RB, that is needed once at the reticle stage, needs a similar load and unload like the wafers. However, reticle RA is needed twice. In between the exposures using reticle RA, it is not necessary to go all the way back to the pod: the reticle can stay on an elevator. Therefore, the total load and unload sequences for reticles are split in two. The 'load reticle' and 'unload reticle' sequences consist of only one step: 'E2S' and 'S2E', respectively. Like 'load wafer' and 'unload wafer' they must be added for the first and the last exposure step in which the reticle is needed, respectively. To transport reticles from the pod to the elevators and back, the 'preload reticle' and 'post unload reticle' sequences are added for the first and the last time the reticle is needed at an elevator, respectively. Like in the wafer case, the first nodes of the reticle preload and

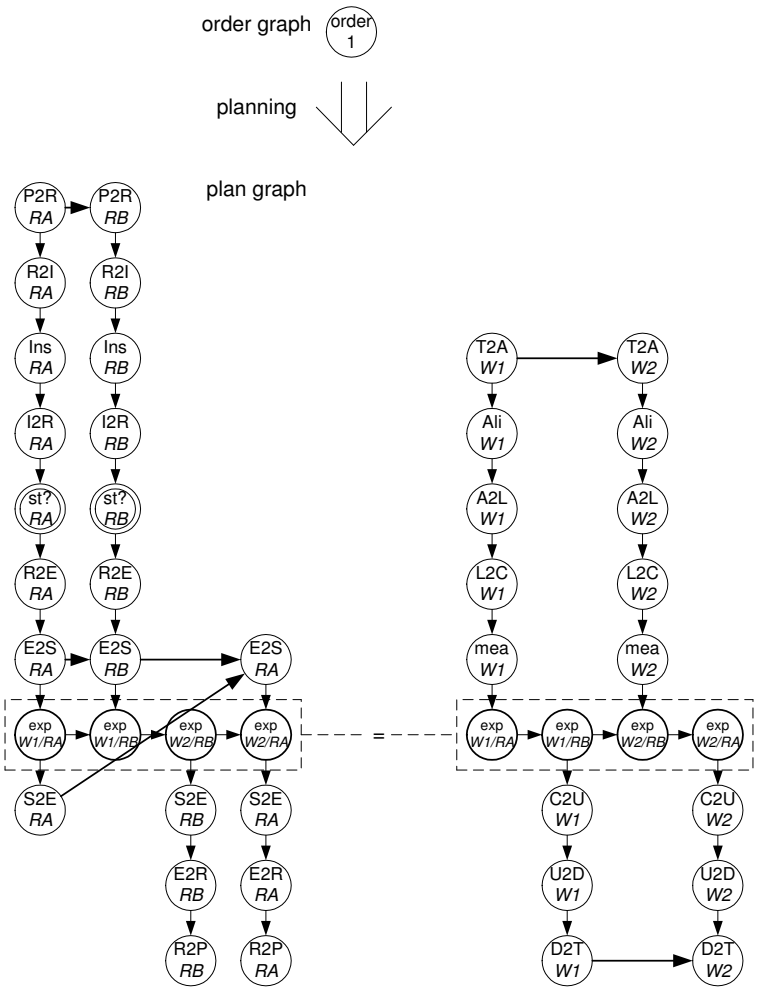


Figure 4.5: Planning: from order graph to task graph

load sequences are connected by a precedence edge.

Analyzing the planning steps of this example, some recurrent types of graph construction steps can be distinguished. At several points, a subgraph ('load wafer', etc.) is inserted into the existing graph. These subgraphs can be regarded as building blocks. The primary manufacturing process, the exposure sequence, forms an important reference to decide whether a subgraph should be added. Instead of inserting the subgraphs in one step, it can be convenient to first insert a single node standing for the entire building block, and then to replace this node by the subgraph itself. In fact, some of the nodes in Fig. 4.5 need to be expanded to a smaller grain size before they can be executed by mechatronic systems. Examples of those nodes are the measure and exposure nodes, that consist of multiple elementary tasks, e.g. to measure one single mark or to expose one single die. For the purpose of this chapter this smaller grain size is not explained. The drill down of nodes into nodes of a smaller grain size by replacing them by building blocks is another recurrent type of planning step. Also some recipe-dependent steps can be distinguished. In the example, the generation of the sequence of exposure steps is one of those steps. Other examples can be found at the lower node grain size: the number of marks to measure or dies to expose is also recipe dependent. However, also here building blocks can be distinguished: exposure of a wafer with a reticle, measurement of a single mark or exposure of a single IC (die). A final recurrent type of planning step is

precedence linking of nodes. For example, linking of sublives of some material instance (e.g. reticle RA), or linking nodes of the same behavior type (e.g. 'T2A'). All planning or graph construction steps are executed only if the system state and the graph fulfill certain criteria. The sequence of construction steps including the applicable criteria can be regarded as planning rules.

4.3.2 Planning functions

In this subsection, a set of generic graph construction functions is defined. The *replaceall* function replaces all nodes in a precedence graph that fulfill some criteria by a subgraph. The nodes that are replaced are called 'parent' nodes, whereas the nodes of the subgraph are called 'child' nodes. The *insertall* function inserts a subgraph at all nodes that fulfill some criteria, called 'foster nodes'. The nodes in the inserted subgraph are called 'orphan' nodes. Furthermore, linking functions are defined to introduce precedence edges: *linkmat* to instantiate a precedence edge between sublives of material instances, and *linkbeh* to instantiate precedence edges between nodes of the same behavior type. As an example, a recipe-dependent generation function is described in the appendix. The domain-specific check functions to determine for instance whether a material instance is needed later on in the exposure sequence are not included in this chapter. Such check functions require information concerning the graph constructed up to then, as well as the state of the system.

The system state, S, consists of 4 components that are not explained further:

- Finish time of the last task per resource
- Material configuration per resource
- Physical state per resource
- Manufacturing state per material instance

During graph construction, a slightly modified TRS definition $\mathcal{D}_{2c}$ is used. Differences with the instantiated, unselected system definition $\mathcal{D}_2$ are:

- I_{2c}, Cb_{2c}, Ce_{2c} are defined for nodes instead of for tasks.

Furthermore, for tracking of the construction process, two elements are introduced.

- Nr_{2c}: $\mathcal{N}_2 \rightarrow \mathcal{P}(\mathcal{N}_2)$ is an additional element giving parent-child relations between nodes.
- Ni_{2c}: $\mathcal{N}_2 \rightarrow \mathcal{P}(\mathcal{N}_2)$ is an additional element giving foster-orphan relations between nodes.

The plan rules need the behavior type of the nodes, e.g. 'T2A', 'exp', 'load wafer' . To define this, the following elements are introduced:

- $\mathcal{B}$ is the set of all possible behavior types
- Nb_{2c}: $\mathcal{N}_2 \rightarrow \mathcal{B}$ gives for each node its behavior type.

Besides this, nodes carry some recipe parameters. From these parameters, building blocks can be generated and material instances can be instantiated.

- Np_{2c}: $\mathcal{N}_2 \rightarrow Pm$ is an additional element giving the parameters of the nodes.

Summarizing, the work dependent part of $\mathcal{D}_{2c}$ is the set of all possible elements from the 17-tuple:
$(\mathcal{N}_{2c}, \mathcal{L}_{2c}, Ln_{2c}, \mathcal{G}_{2c}, Gn_{2c}, Ga_{2c}, I_{2c}, Sb_{2c}, Se_{2c}, Cb_{2c}, Ce_{2c}, P_{2c}, Pt_{2c}, Nr_{2c}, Ni_{2c}, Nb_{2c}, Np_{2c})$

The starting point of the planning process is a construction system definition $\mathcal{D}_{2c}$ containing the orders to plan as nodes in a precedence graph. This can be a sequence of order nodes representing an order queue, but if there are no priorities order nodes can also be modelled in parallel. After application of the planning rules, a more detailed system definition $\mathcal{D}_{2c}$ results, that is converted into an instantiated unselected system definition $\mathcal{D}_2$. In this conversion, the information that is not needed in $\mathcal{D}_2$ is removed from $\mathcal{D}_{2c}$. This concerns parent and foster node related elements and node behaviors. Furthermore, all nodes that are not clusters or groups become tasks. As this conversion is straightforward, the conversion function is not described in this chapter.

To determine to which nodes a construction step must be applied, the following functions are involved. Set C: $\mathcal{P}(\mathcal{N}_{2c} \times \mathcal{D}_{2c} \times S) \rightarrow \mathbb{B}$) is a library of check functions c_x that determine for some node given some system definition and system state whether certain criteria hold. Function *nodestobehandled*: $\mathcal{D}_{2c} \times \mathcal{P}(\mathcal{B}) \times \mathcal{P}(C) \times S \rightarrow \mathcal{P}(\mathcal{N}_{2c})$ is a function that determines which nodes are to be handled. In case they are used in a replace function, these nodes are called parent nodes, whereas these nodes are called foster nodes in case they are used in an insert function. The nodes to be handled are the nodes in an existing system definition D^e_{2c} with a behavior that is in a set of behaviors br and for which condition checks cr hold in some system state $sstate$:

$$nodestobehandled(D^e_{2c}, br, cr, sstate) = \{n^e | n^e \in \mathcal{N}^e_{2c} \wedge Nb(n^e) \in br \wedge (\forall c : c \in cr : c(n^e, D^e_{2c}, sstate))\} \quad (4.1)$$

For generation of the exposure sequence, the node parameters need to contain part of the recipe information. For the purpose of this chapter, the node parameters are a tuple of tuples containing a set of capabilities and a list of material instance sets Np_{2c}: $\mathcal{N}_2 \rightarrow (\mathcal{P}(\mathcal{C}) \times \mathcal{P}(\mathcal{M})^*)^2$. By convention, the first element of the tuple concerns wafers and the second element concerns reticles. Furthermore, the sets of capabilities involved in wafer and reticle processing are assumed to be disjoint. For the wafers as well as the reticles, the capabilities involved in this type of material and the material instances themselves are described. In the appendix, a function that generates an exposure sequence using the node parameters, *genes*, is described.

The nodes in a predefined basic building block are uninstantiated, implying that they have no material assigned to them. The nodes in a generated building block, that is generated using a basic building block, do have material assigned to them. The information required for that is obtained from the parameters of the parent node of the generated building block. In the actual replacement step, the material assigned to the nodes in the

generated building block should be unchanged. In case of replacing a node by a basic building block or inserting a basic building block, material should be inherited from the parent or foster node. To assign material to a building block node, a function *matassign* is defined with a parameter m to define whether or not material should be obtained from the parameters *np* of a parent or foster node. Other parameters are the begin and the end material configuration of the building block node, *cba* and *cbe*, respectively.

Function *matassign*: $\mathbb{B} \times \mathcal{P}(\mathcal{C} \times \mathcal{P}(\mathcal{M})) \times \mathcal{P}(\mathcal{C} \times \mathcal{P}(\mathcal{M})) \times (\mathcal{P}(\mathcal{C}) \times \mathcal{P}(\mathcal{M})^*)^2 \rightarrow \mathcal{P}(\mathcal{C} \times \mathcal{P}(\mathcal{M})) \times \mathcal{P}(\mathcal{C} \times \mathcal{P}(\mathcal{M}))$ is defined as follows:

$matassign(m, cba, cea, np) =$

$$\begin{cases} (cba, cea) & \text{if } \neg m \\ \begin{array}{ll} (& \{(x.0, hd(y.1)) | x \in cba \wedge x.1 \neq \varnothing \wedge y \in \{np.0, np.1\} \wedge x.0 \in y.0\} \\ & \cup \{x | x \in cba \wedge x.1 = \varnothing\} \\ , & \{(x.0, hd(y.1)) | x \in cea \wedge x.1 \neq \varnothing \wedge y \in \{np.0, np.1\} \wedge x.0 \in y.0\} \\ & \cup \{x | x \in cea \wedge x.1 = \varnothing\} \\ ) & \end{array} & \text{if } m \end{cases} \tag{4.2}$$

In case materials are not inherited, the begin and the end material configurations of a new node are copied from the building block called addition. In case materials are inherited and (default) material is configured in the node of the addition, the set of material instances is inherited from the element of the parent node parameter that matches the involved capability, by taking the first element from the list. If no materials are configured in a node of the addition, this remains the same for the resulting instantiated material configuration of that node.

Using the functions defined above, the functions to replace nodes by building blocks and to insert building blocks can be defined.

Let function *replaceone*: $\mathcal{D}_{2c} \times \mathcal{N}_{2c} \times \mathcal{D}_{2c} \times \mathbb{B} \rightarrow \mathcal{D}_{2c}$ be a function that replaces in an existing system definition D^e_{2c} a parent node n^e by an addition D^a_{2c}, taking into account whether involved materials either or not must be inherited (depending on the last parameter m). In the sequel, it is assumed that the nodes in the existing system definition D^e_{2c} do not intersect the nodes in the addition D^a_{2c}: $\mathcal{N}^e_{2c} \cap \mathcal{N}^a_{2c} = \varnothing$. This might imply renaming of nodes in the addition.

Function *replaceone* can be defined as follows:

$replaceone(D^e_{2c}, n^e, D^a_{2c}, m) = D^{e\prime}_{2c}$ such that $D^{e\prime}_{2c} = D^e_{2c} \cup D^a_{2c}$ where $D^e_{2c} \cup D^a_{2c}$ is a pairwise union of all set definition elements[1] except that:

$$\begin{array}{lll} (\forall n^{e\prime} & : & n^{e\prime} \in \mathcal{N}^a_{2c} \\ & : & (Cb^{e\prime}_{2c}(n^{e\prime}), Ce^{e\prime}_{2c}(n^{e\prime})) = matassign(m, Cb^a_{2c}(n^{e\prime}), Ce^a_{2c}(n^{e\prime}), Np^e_{2c}(n^e)) \\ & \wedge & ((\nexists n^a : n^a \in \mathcal{N}^a_{2c} : n^a \in anc(n^{e\prime}) \vee (n^a, n^{e\prime}) \in P^a_{2c}) \\ & & \Rightarrow (\forall n : (n, n^e) \in P^e_{2c} : (n, n^{e\prime}) \in P^{e\prime}_{2c} \wedge (n, n^e) \notin P^{e\prime}_{2c}) \\ & & \wedge (\forall n : (n, n^e) \in Pt^e_{2c} : (n, n^{e\prime}) \in Pt^{e\prime}_{2c} \wedge (n, n^e) \notin Pt^{e\prime}_{2c})) \\ & \wedge & ((\nexists n^a : n^a \in \mathcal{N}^a_{2c} : n^a \in anc(n^{e\prime}) \vee (n^{e\prime}, n^a) \in P^a_{2c}) \\ & & \Rightarrow (\forall n : (n^e, n) \in P^e_{2c} : (n^{e\prime}, n) \in P^{e\prime}_{2c} \wedge (n^e, n) \notin P^{e\prime}_{2c}) \\ & & \wedge (\forall n : (n^e, n) \in Pt^e_{2c} : (n^{e\prime}, n) \in Pt^{e\prime}_{2c} \wedge (n^e, n) \notin Pt^{e\prime}_{2c})) \\ & \wedge & Np^{e\prime}_{2c}(n^{e\prime}) = Np^e_{2c}(n^e) \\ & \wedge & n^{e\prime} \in Nr^{e\prime}_{2c}(n^e) \\ ) & & \end{array} \tag{4.3}$$

[1] As $\mathcal{N}^e_{2c} \cup \mathcal{N}^a_{2c}$, this can be defined similarly for definition elements which are functions.

The system definition that results from function *replaceone* is the same as the union of the existing system definition and the addition, except for the assignment of involved material and the (strong) inheritance of precedence relations and parameters from the replaced parent node. The assignment of involved material is taken care of by function *matassign*. Concerning precedence relations, all top front nodes in the addition (i.e. at the top of the node hierarchy and the front of the precedence graph) inherit the (tied) precedence edges to the parent node, whereas all top rear nodes inherit the (tied) precedence edges from the parent node. The precedence edges to and from the parent node are deleted. Note that weak precedence inheritance can be useful, but this is not explained in this chapter. Furthermore, the nodes added are instantiated as children of the parent node.

Using this, function *replaceall*: $\mathcal{D}_{2c} \times \mathcal{P}(\mathcal{N}_{2c}) \times \mathcal{D}_{2c} \times \mathbb{B} \rightarrow \mathcal{D}_{2c}$ can be defined recursively:

$replaceall(D^e_{2c}, N^{er}, D^a_{2c}, m) =$

$$\begin{cases} D^e_{2c} & \text{if } N^{er} = \varnothing \\ replaceall(replaceone(D^e_{2c}, n^{er}, D^a_{2c}, m), N^{er} \backslash \{n^{er}\}, D^a_{2c}, m) & \text{if } N^{er} \neq \varnothing \\ & \quad \wedge\, n^{er} \in N^{er} \end{cases} \tag{4.4}$$

To replace all nodes in an existing system definition D^e_{2c} with a behavior that is in a set of behaviors br and for which condition checks cr hold in some system state by some addition D^a_{2c}, whether or not inheriting involved materials (depending on m), the following expression can be used: $replaceall(D^e_{2c}, nodestobehandled(D^e_{2c}, br, cr, systemstate), D^a_{2c}, m)$.

Let function *insertone*: $\mathcal{D}_{2c} \times \mathcal{N}_{2c} \times \mathcal{D}_{2c} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathcal{P}(\mathcal{B}) \rightarrow \mathcal{D}_{2c}$ be a function that inserts in an existing system definition D^e_{2c} before or after a foster node n^e (depending on b, see system definition below) an addition D^a_{2c}. It takes into account whether involved materials must be inherited from n^e (depending on m), and whether the precedence relation to the foster must be tied (depending on *tied*). The addition is inserted in between the foster node and the nodes preceding or succeeding the foster node (depending on b) if their behavior is in *or*: the 'opposite' nodes. Function *insertone* can be defined as follows:

$insertone(D^e_{2c}, n^e, D^a_{2c}, m, b, tied, or) = D^{e\prime}_{2c}$ such that $D^{e\prime}_{2c} = D^e_{2c} \cup D^a_{2c}$ except that:

$$\begin{aligned}
&(\forall n^{e\prime} : n^{e\prime} \in \mathcal{N}^a_{2c} \\
&\quad : (Cb^{e\prime}_{2c}(n^{e\prime}), Ce^{e\prime}_{2c}(n^{e\prime})) = matassign(m, Cb^a_{2c}(n^{e\prime}), Ce^a_{2c}(n^{e\prime}), Np^e_{2c}(n^e)) \\
&\quad \wedge (b \wedge (\nexists n^a : n^a \in \mathcal{N}^a_{2c} : n^a \in anc(n^{e\prime}) \vee (n^{e\prime}, n^a) \in P^a_{2c}) \\
&\qquad \Rightarrow (n^{e\prime}, n^e) \in P^{e\prime}_{2c} \wedge (tied \Rightarrow (n^{e\prime}, n^e) \in Pt^{e\prime}_{2c})) \\
&\quad \wedge (b \wedge (\nexists n^a : n^a \in \mathcal{N}^a_{2c} : n^a \in anc(n^{e\prime}) \vee (n^a, n^{e\prime}) \in P^a_{2c}) \\
&\qquad \Rightarrow (\forall n : (n, n^e) \in P^e_{2c} \wedge Np^e_{2c}(n) \in or : (n, n^{e\prime}) \in P^{e\prime}_{2c} \wedge (n, n^e) \notin P^{e\prime}_{2c}) \\
&\qquad \wedge (\forall n : (n, n^e) \in Pt^e_{2c} \wedge Np^e_{2c}(n) \in or : (n, n^{e\prime}) \in Pt^{e\prime}_{2c} \wedge (n, n^e) \notin Pt^{e\prime}_{2c})) \\
&\quad \wedge (\neg b \wedge (\nexists n^a : n^a \in \mathcal{N}^a_{2c} : n^a \in anc(n^{e\prime}) \vee (n^a, n^{e\prime}) \in P^a_{2c}) \\
&\qquad \Rightarrow (n^e, n^{e\prime}) \in P^{e\prime}_{2c} \wedge (tied \Rightarrow (n^e, n^{e\prime}) \in Pt^{e\prime}_{2c})) \\
&\quad \wedge (\neg b \wedge (\nexists n^a : n^a \in \mathcal{N}^a_{2c} : n^a \in anc(n^{e\prime}) \vee (n^{e\prime}, n^a) \in P^a_{2c}) \\
&\qquad \Rightarrow (\forall n : (n^e, n) \in P^e_{2c} \wedge Np^e_{2c}(n) \in or : (n^{e\prime}, n) \in P^{e\prime}_{2c} \wedge (n^e, n) \notin P^{e\prime}_{2c}) \\
&\qquad \wedge (\forall n : (n^e, n) \in Pt^e_{2c} \wedge Np^e_{2c}(n) \in or : (n^{e\prime}, n) \in Pt^{e\prime}_{2c} \wedge (n^e, n) \notin Pt^{e\prime}_{2c})) \\
&\quad \wedge Np^{e\prime}_{2c}(n^{e\prime}) = Np^e_{2c}(n^e) \\
&\quad \wedge n^{e\prime} \in Ni^{e\prime}_{2c}(n^e) \\
&)
\end{aligned} \tag{4.5}$$

Function *insertone* resembles *replaceone*, except for the precedence relations. In case the insertion is done before the foster node, all top rear nodes of the addition get a (tied

if applicable) precedence edge to the foster node. Furthermore, precedence edges are inherited from the opposite nodes to all top front nodes of the addition. In case the insertion is done after (not before) the foster node, the precedence edges are instantiated the other way around.

Using this, function *insertall*: $\mathcal{D}_{2c} \times \mathcal{P}(\mathcal{N}_{2c}) \times \mathcal{D}_{2c} \times \mathbb{B} \times \mathbb{B} \times \mathcal{P}(\mathcal{B} \times \mathbb{B}) \times \mathcal{P}(\mathcal{B}) \rightarrow \mathcal{D}_{2c}$ can be defined recursively:

$insertall(D^e_{2c}, N^{ei}, D^a_{2c}, m, b, brt, or) =$

$$\begin{cases} D^e_{2c} & \text{if} \quad N^{er} = \varnothing \\ insertall(insertone(D^e_{2c}, n^{ei}, D^a_{2c}, m, b, bt.1, or) & \text{if} \quad N^{er} \neq \varnothing \\ \qquad , N^{ei} \backslash \{n^{ei}\}, D^a_{2c}, m, b, brt, or & \qquad \wedge\ n^{ei} \in N^{ei} \\ \qquad) & \qquad \wedge\ bt \in brt \\ & \qquad \wedge\ bt.0 = Nb^e_{2c}(n^{ei}) \end{cases} \tag{4.6}$$

Here the tuples in $brt \subseteq \mathcal{B} \times \mathbb{B}$ define the behaviors of the foster nodes and whether or not the precedence relation to the foster node must be tied.

To insert some addition D^a_{2c} in an existing system definition D^e_{2c} before or after (depending on b) all foster nodes, whether or not inheriting involved materials (depending on m), and whether or not in between opposite nodes with behavior in *or* preceding or succeeding the foster nodes, the following expression can be used:

$insertall(D^e_{2c}, nodestobehandled(D^e_{2c}, \{bt.0 | bt \in brt\}, cr, systemstate), D^a_{2c}, m, b, brt, or)$.

To link the sublives of material instances, first the sublives are extracted from the system definition using function P_{2m}: $\mathcal{D}_{2c} \times \mathcal{M}_{2c} \rightarrow \mathcal{P}(\mathcal{N}_{2c} \times \mathcal{N}_{2c})$. Function P_{2m} is a function describing for a material $m \in \mathcal{M}_{2c}$ in a TRS definition $D^e_{2c} \in \mathcal{D}_{2c}$, a precedence relation between related nodes without redundant edges:

$P_{2m}(D^e_{2c}, m) =$

$$\begin{array}{l} \{(n, n') \\ \mid (n, n') \in P^e_{2c} \\ \wedge \{cm.0 | cm \in Ce_{2c}(n), m \in cm.1\} = \{cm.0 | cm \in Cb_{2c}(n'), m \in cm.1\} \\ \wedge \neg redundant(n, n', P^e_{2c}) \\ \} \end{array} \tag{4.7}$$

Here function *redundant*: $\mathcal{N}_{2c} \times \mathcal{N}_{2c} \times P_{2c} \rightarrow \mathbb{B}$ determines whether a precedence edge (n, n') is redundant in a precedence relation P:

$$redundant(n, n', P) = (\exists n'' : n'' \in \mathcal{N}_{2c}, n'' \neq n, n'' \neq n' : path(n, n'', P) \wedge path(n'', n', P)) \tag{4.8}$$

Function *path*: $\mathcal{N}_{2c} \times \mathcal{N}_{2c} \times P_{2c} \rightarrow \mathbb{B}$ used above determines whether there is a path between two nodes n and n' in a precedence relation P:

$path(n, n', P) =$

$$\begin{cases} true & \text{if} \quad n \in \{n'\} \cup anc(n') \\ (\exists n'', n''' : n''' \in \{n\} \cup anc(n), (n''', n'') \in P & \text{if} \quad n \notin \{n'\} \cup anc(n') \\ \qquad : path(n'', n', P) & \\ ) & \end{cases} \tag{4.9}$$

Here, ancestor function $anc : N \rightarrow \mathcal{P}(N)$ gives the nodes in which a certain node is contained. The set $anc(n)$ is the smallest set satisfying the following conditions:

- if $n \in G_n(n')$ or $n \in L_n(n')$, then $n \in anc(n)$;
- if $n' \in anc(n)$ and $n'' \in anc(n')$ then $n'' \in anc(n)$.

Function *linkmat*: $\mathcal{D}_{2c} \times \mathcal{B} \rightarrow \mathcal{D}_{2c}$ is a function that links the sub lives of material instances together in the same order as the associated nodes of primary behavior $pb \in \mathcal{B}$.

$linkmat(D^e_{2c}, pb) = D^{e\prime}_{2c}$ such that $D^{e\prime}_{2c} = D^e_{2c}$ except that:

$$
\begin{array}{lll}
(\forall m, n, n' & : & m \in M^e_{2c}, n, n' \in N^e_{2c} \\
& , & (\nexists n'' : n'' \in N^e_{2c} : (n, n'') \in P_{2m}(D^e_{2c}, m) \vee (n'', n') \in P_{2m}(D^e_{2c}, m)) \\
& , & (\exists n'', n''' : n'', n''' \in N^e_{2c}, Nb^e_{2c}(n'') = pb, Nb^e_{2c}(n''') = pb \\
& & : path(n'', n''', P^e_{2c}) \wedge path(n'', n, P^e_{2c}) \wedge path(n', n''', P^e_{2c}) \\
& &) \\
& : & (n, n') \in P^{e\prime}_{2c} \\
)
\end{array}
\tag{4.10}
$$

Function *linkbeh*: $\mathcal{D}_{2c} \times \mathcal{P}(\mathcal{B} \times \mathbb{B}) \times \mathcal{B} \rightarrow \mathcal{D}_{2c}$ is a function that links tasks with a certain behavior together in the same order as the associated nodes of primary behavior $pb \in \mathcal{B}$. The function takes the behaviors in the first elements of the tuples in set $bpr \in \mathcal{P}(\mathcal{B} \times \mathbb{B})$ into account, where the second element of each tuple indicates whether the behavior concerns pre-processing or post-processing (true or false, respectively).

$linkbeh(D^e_{2c}, bpr, pb) = D^{e\prime}_{2c}$ such that $D^{e\prime}_{2c} = D^e_{2c}$ except that:

$$
\begin{array}{lll}
(\forall b, p, n, n' : & & (b, p) \in bpr, n, n' \in N^e_{2c}, Nb^e_{2c}(n) = b, Nb^e_{2c}(n') = b \\
& , \ (& \exists n'', n''' \\
& : & n'', n''' \in N^e_{2c}, Nb^e_{2c}(n'') = pb, Nb^e_{2c}(n''') = cb, path(n'', n''', P^e_{2c}) \\
& : & (p \wedge path(n, n'', P^e_{2c}) \wedge path(n', n''', P^e_{2c})) \\
& & \vee (\neg p \wedge path(n'', n, P^e_{2c}) \wedge path(n''', n', P^e_{2c})) \\
& &) \\
&) & \\
& : & (n, n') \in P^{e\prime}_{2c} \\
)
\end{array}
\tag{4.11}
$$

4.3.3 Automatic planning of the typical example

The planning functions described in the previous subsection enable automatic planning. In this subsection, automatic planning is illustrated for the typical example order 1.

Planning involves application of the following sequence of planning rules:

1. First, the order node is replaced by an exposure sequence that is generated from the parameters of the order node using function *genes*.

   The function call looks as follows: $replaceall(D^e_{2c}, nodestobehandled(D^e_{2c}, \{n1\}, \{c_0\}), genes(D^a_{2c}, false, Np(n1), 2, 2, D^\epsilon_{2c})), false)$

   Here:

   - D^e_{2c} is the existing system definition, containing the node of order 1, $n1$ at that moment.
   - c_0 is a dummy check function that always returns `true`.
   - D^a_{2c} is the system definition of the building block containing one exposure node.

- D_{2c}^{ϵ} is the empty system definition.

2. Then, a 'load wafer' is inserted before each node of behavior 'exp' that fulfills the following criteria:

   (a) not preceded by a 'mea' node yet, and

   (b) not preceded by a node of behavior 'exp' involving the same wafer, and

   (c) the wafer is not present at a chuck yet.

   The function call looks as follows:

   $insertall(D_{2c}^{e}, nodestobehandled(D_{2c}^{e}, \{'exp'\}, \{c_a, c_b, c_c\}, systemstate),$
   $D_{2c}^{`load\_wafer'}, true, true, \{('exp', false)\}, \{\})$

   Here:

   - D_{2c}^{e} is the existing system definition, the result of planning step 1
   - c_a, c_b, c_c are the check functions implementing criteria a through c listed above

3. An 'unload wafer' is inserted after each node of behavior 'exp' that fulfills the following criteria:

   (a) not succeeded by a 'C2U' node yet, and

   (b) not succeeded by a node of behavior 'exp' involving the same wafer.

4. A 'load reticle' is inserted before each node of behavior 'exp' that fulfills the following criteria:

   (a) not preceded by a 'E2S' yet, and

   (b) not preceded by a node of behavior 'exp' involving the same reticle, and

   (c) the reticle is not present at the stage yet.

5. An 'unload reticle' is inserted after each node of behavior 'exp' that fulfills the following criteria:

   (a) not succeeded by a 'E2S' yet, and

   (b) not succeeded by a node of behavior 'exp' involving the same reticle.

6. A 'preload reticle' is inserted before each node of behavior 'E2S' that fulfills the following criteria:

   (a) not preceded by a 'R2E' yet, and

   (b) the reticle is not left at an elevator after an earlier expose and reticle unload, and

   (c) the reticle is not present at an elevator yet.

7. A 'post unload reticle' is inserted after each node of behavior 'S2E' that fulfills the following criteria:

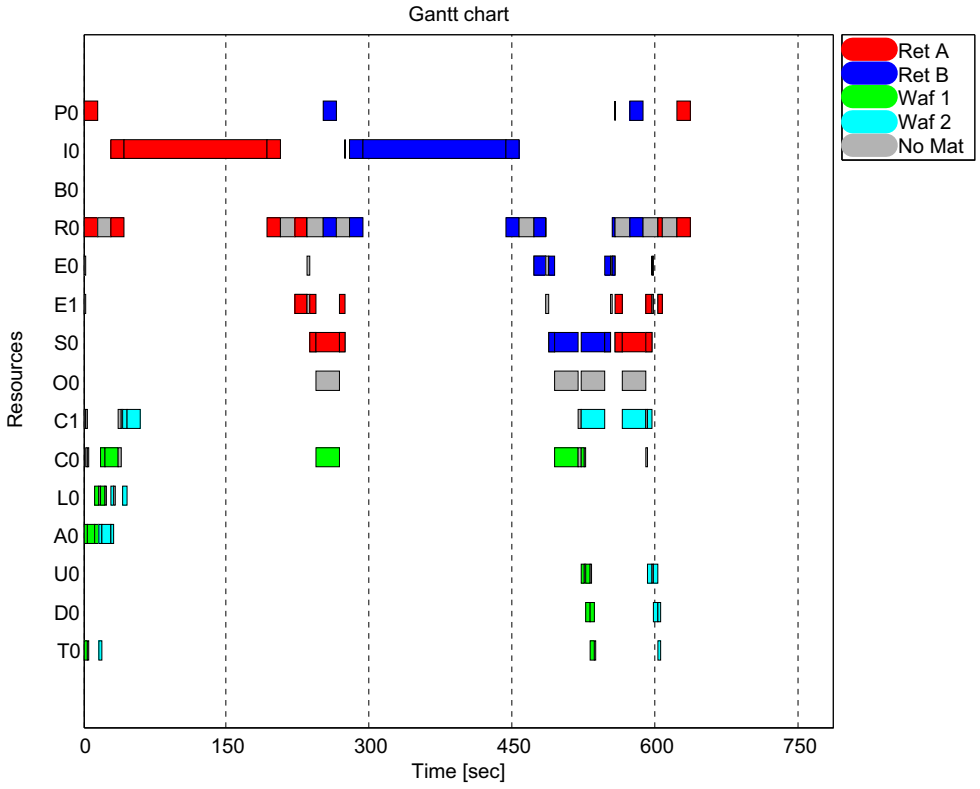


Figure 4.6: Schedule for typical order 1

(a) not succeeded by a 'E2R' yet, and

(b) not needed for a later 'exp', and

(c) the reticle is not present at an elevator yet.

Note: these rules are sufficient for this example, and the other examples in this chapter. To avoid violation of the material capacity of the elevators in all situations, additional rules for reticle pre loads and post unloads should be added.

8. The material sublives are linked together around the exposure sequence using $linkmat(D^e_{2c}, 'exp')$.

9. Precedence edges between consecutive nodes of the same behavior are added using $linkbeh(D^e_{2c}, \{('P2R', \texttt{true}), ('E2S', \texttt{true}), ('T2A', \texttt{true}), ('D2T', \texttt{false})\})$.

10. The construction system definition is converted into an instantiated unselected system definition.

The result of these ten planning steps matches Fig. 4.5. In Fig. 4.6, a schedule for order 1 is shown.

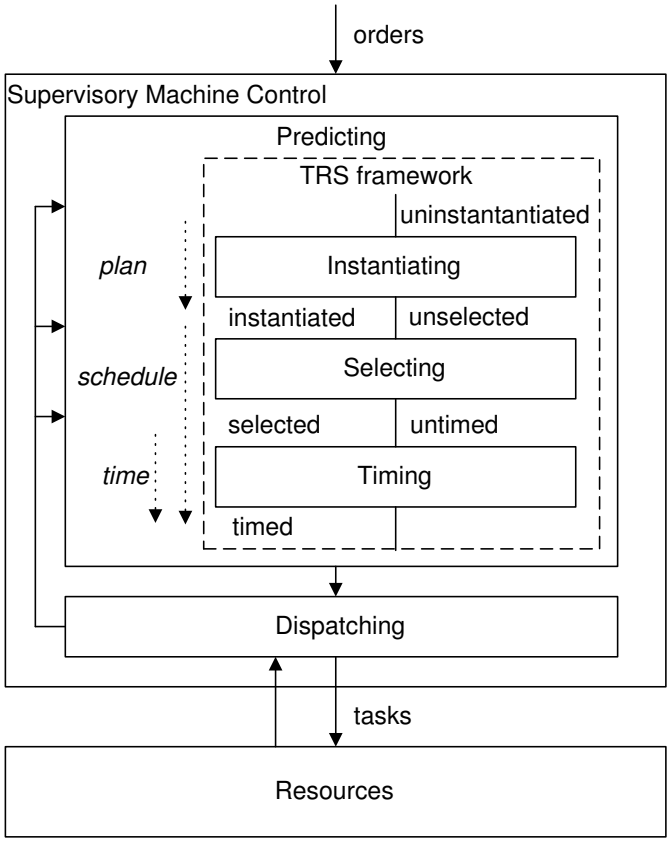


Figure 4.7: SMC reaction functionality framework

4.4 Reaction to triggers

Whereas Chapters 2 and 3 ([11, 12]) and the previous section describe the transformation functionalities of the layered TRS framework depicted in Fig. 4.1, this section describes how the TRS framework can be embedded in SMC. Such an SMC framework is shown in Fig. 4.7. SMC must react to triggers from its environment. These triggers can originate from the user as well as from the machine resources (feedback) and may require revision of the schedule. In the first subsection, the involved functionalities and definitions of the SMC framework are summarized. The other subsections describe the reaction to several types of triggers. Two categories of triggers originate from the machine resources: current work delays, and exceptions occurring. The other category originates from the user: new work arrives. Reaction implies translation of the triggers into a revised schedule, using the functionalities and definitions of the first subsection.

4.4.1 Framework

The SMC framework consists of two parts: predicting and dispatching. The predicting part accomodates the predictive scheduling functionality, that translates triggers into schedules. The dispatching part is the real-time part, that connects the predictive part to the resources. The main functionality of this part is dispatching the scheduled tasks to the resources. In the predictive part, the layered TRS framework is embedded (see the dashed box), which is implemented by three transformation functions: *plan*, *schedule*, and *time*. The *plan* function is explained in the previous section and implements the instanti-

ating transformation. The *time* function implements the timing transformation, whereas the *schedule* function implements both the selecting and the timing transformations, as explained in Chapters 2 and 3 ([11, 12]).

An important requirement is avoidance of control overhead. Therefore, reaction or predicting activity should take place in the shadow of real-time activity if possible. For this reason, the part of the schedule being executed in the dispatcher that can run in parallel with reaction activities to adapt the schedule is not revised. When following the complete trajectory from the arrival of an order up to the moment at which the work is finished, the following phases of the work are distinguished:

- Work Ordered, WO.
- Work Planned, WP.
- Work Scheduled, WS.
- Work to be Dispatched, WD: this is the part of the work that is not yet in progress, but is scheduled to be in progress (safe bound) after reaction activity (including ties). Reaction takes place in the (time) shadow of the execution of WD.
- Work In progress, WI.
- Work Executed, WE.
- Work Finished, WF: this is the part of the work executed that does not matter anymore: all successors of this part of the work are executed. This 'history' is not required for control anymore and can be removed.

A scheduled task ($\mathcal{D}_0$) can be in WS, WD, WI, WE, or WF, whereas nodes in a plan ($\mathcal{D}_2, \mathcal{D}_{2c}$) can also be in WO or WP. By convention, the phases concerned in part of a definition are denoted in superscript, e.g. $\mathcal{D}_0^{id}$ denotes the part of a schedule (0 in subscript) that is in process or will be dispatched (*id* in superscript).

In this section, we use a *plan* function without any configuration parameters, thus abbreviating the functionality of the previous section: $plan(initstate, D_{2c}) = D_2^p$. Furthermore, we use a *schedule* and a *time* function that have a history schedule D_0^h as a parameter. This history schedule is needed to avoid violation of precedence relations crossing the initial state (time contour): $schedule(D_0^h, initstate, D_2^t, D_2) = (D_0^s, D_1^s, endstate)$. Here, D_2 is the (unselected) definition of the work to be scheduled, and D_2^t is the total definition concerning both the history schedule and the work to be scheduled: $time(D_0^h, initstate, D_1^t, D_1) = (D_0^s, endstate)$. Here D_1 is the (selected) definition of the work to be timed, and D_1^t is the total selection concerning both the history schedule and the work to be timed.

Moreover, we need partitions of total system definitions as parameters for the *plan*, *time* and *schedule* functions, depending on the required phases. To this end, we use extraction functions to extract from total definitions the partition concerning a certain set of nodes. For the purpose of this chapter, we only give the format of the extraction functions:

- *extract2c* is a function that extracts from an unselected TRS definition D_{2c} the part that is related to a set of nodes N: $extract2c(D_{2c}, N) = D_{2c}^e$.
- *extract2* is a function that extracts from an unselected TRS definition D_2 the part that is related to a set of nodes N: $extract2(D_2, N) = D_2^e$.

- *extract1* is a function that extracts from a selected TRS definition D_1 the part that is related to a set of tasks T: $extract1(D_1, T) = D_1^e$.
- *extract0* is a function that extracts from a timed TRS definition D_0 the part that is related to a set of tasks T: $extract0(D_0, T) = D_0^e$. A TRS definition D_0 consists of two types of tasks: core tasks and setup tasks. Core tasks also exist in the higher TRS definition levels, and setup tasks are added to make sure that the physical resource end states of tasks match the resource start states of consecutive tasks.

4.4.2 Current work delays

Due to several reasons, the actual duration of tasks can differ from the scheduled predicted durations. To be robust for that, three guards are checked before dispatching a task:

1. Is the task start time reached?
2. Are the involved resources idle?
3. Are the preceding tasks finished?

The first guard ensures that there is no effect of tasks taking less time for the rest of the actual execution. However, tasks taking more time than scheduled can cause the scheduled prediction to be out of sync with actual execution. In Fig. 4.8, this effect is illustrated for a delay of the inspection of reticle RA in order 1. The scheduled timing is depicted at the resources with a '_s' extension, whereas the actual timing is depicted at the resources without extension.

The fact that the remaining predicted schedule is out of sync with reality may result in violations of time window constraints [12] or suboptimality. Therefore, the actual finish of the delayed task could form a trigger for revision of the schedule. In Fig. 4.8, the trigger time, t_t, and the (worst case) duration of the reaction, t_r, are depicted with vertical lines. The tasks scheduled to start before the finish of the reaction form the work to be dispatched (WD in Fig. 4.8), which is not revised. The work scheduled (WS in Fig. 4.8) is withdrawn from the dispatcher and is returned after revision.

Two reaction scenarios are defined for this type of trigger originating from the current work:

C1a Update current work, re-time

C1b Update current work, re-schedule

Both scenarios consist of two steps:

1. The purpose of the first step is to get a good starting point for revision of the work scheduled (WS). This is done by re-timing the work up to the work to be dispatched (a), and accounting for the reaction time (b).

   (a) First the work to be re-timed is to be determined, which consists of WI up to WD. The set of timed tasks covering D_0^{id}, T_0^{id}, is determined in three steps:

      i. Determine the core tasks after WI that are scheduled to be in progress after the reaction time, T_2^d $(= T_1^d)$:
      $T_2^d = \{t | t \in \mathcal{T}_2 \backslash (T_0^e \cup T_0^i), \tau_{S_0}(t) < (t_t + t_r)\}$.

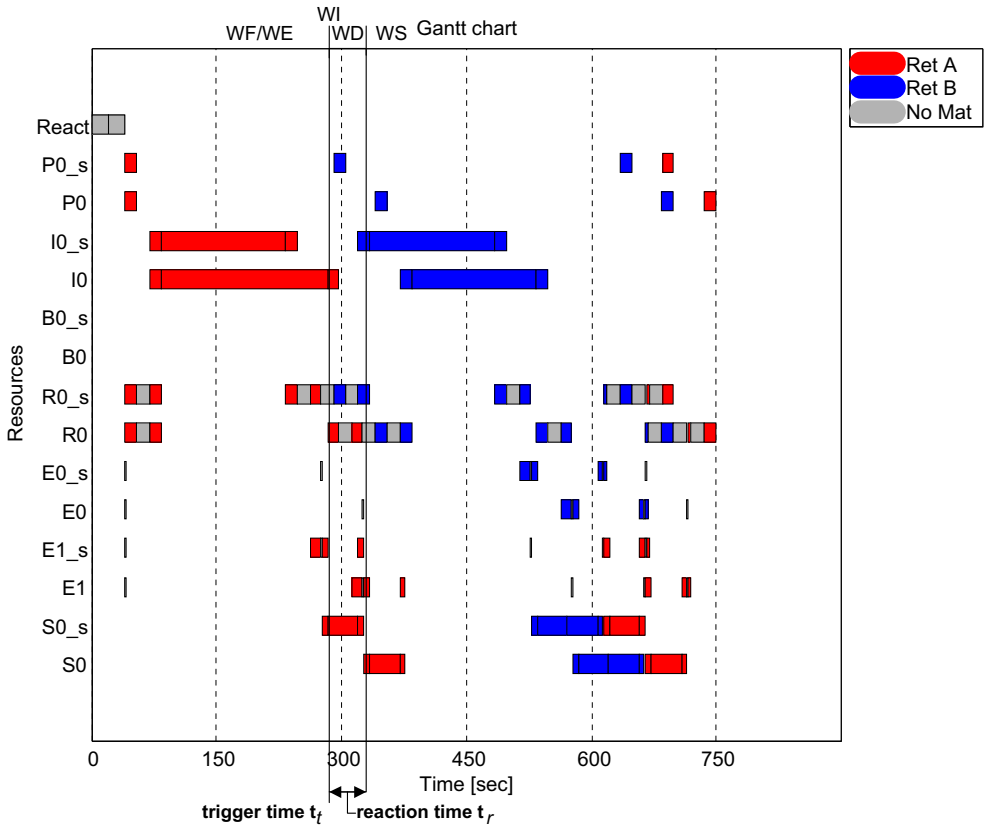


Figure 4.8: Inspection of reticle A delays

ii. Determine the core tasks T_2^t that are tied to T_0^e, T_0^i, or T_2^d:
$T_2^t = \{t | t \in \mathcal{T}_2, (\exists t' : t' \in (T_0^e \cup T_0^i \cup T_2^d) : path(t', t, Pt_2))\}$.

iii. Include the setup tasks T_0^{idt} in between T_0^e, T_0^i, T_2^d, and T_2^t:
$T_0^{idt} = \{t | t \in \mathcal{T}_0, (\exists t', t'' : t' \in (T_0^e \cup T_0^i \cup T_2^d), t'' \in (T_2^d \cup T_2^t) : path(t', t, P_1) \wedge path(t, t'', P_1))\}$.
Then $T_0^{id} = T_0^i \cup T_2^d \cup T_2^t \cup T_0^{idt}$.
With $time(D_0^e, Estate, D_1, D_1^{id})$ the work up to WD is re-timed, where:

- D_0^e= $extract0(D_0, T_0^e)$;
- $Estate$ is the state after WE;
- D_1 contains at least WE till WD (As WS is also allowed no extraction is needed);
- $D_1^{id} = extract1(D_1, T_0^{id})$.

(b) As the work thereafter can never start earlier than $(t_t + t_r)$, the state after this work is to be updated with respect to the finish contour:
$Dstate' = Dstate$ except that $(\forall r : r \in \mathcal{R} : finish(r) = \max(finish(r), t_t + t_r))$.

2. The purpose of the second step is to re-time (C1a) or re-schedule (C1b) the work scheduled.

C1a With $time(D_0^{id}, Dstate', D_1, D_1^s)$ the rest of the current work is re-timed, where:

- D_0^{id}= $extract0(D_0, T_0^{id})$;
- $Dstate'$ is the state after WD including reaction time, resulting from step 1;

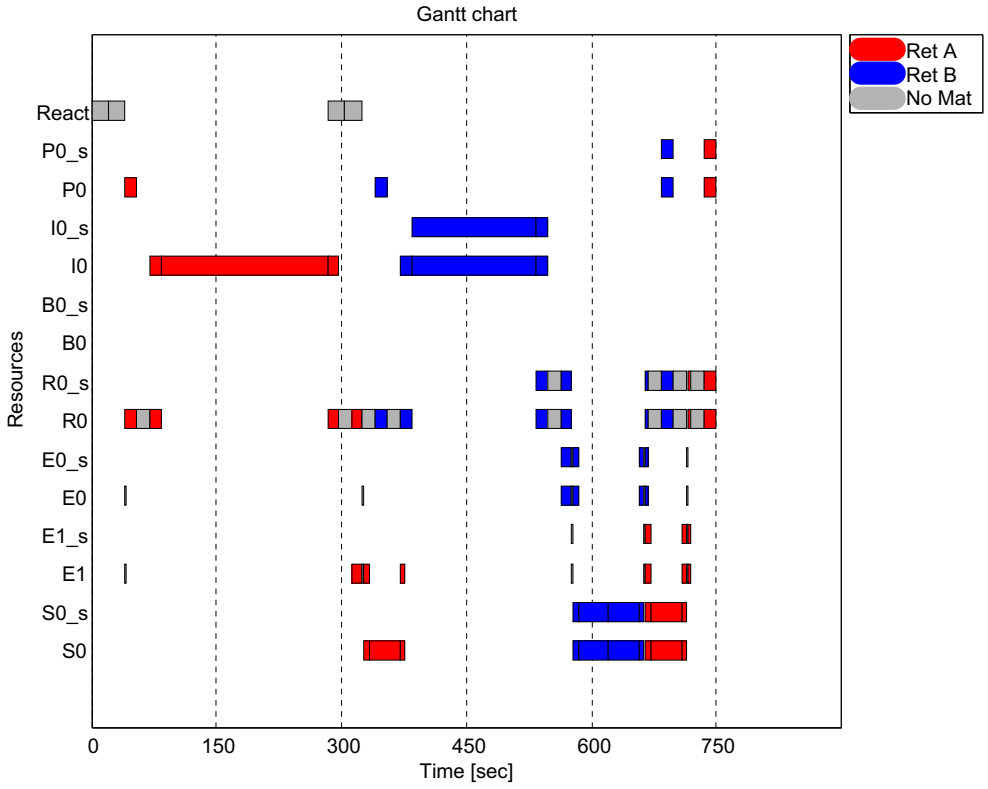


Figure 4.9: Updated schedule after delay

- $D_1^s = extract1(D_1, (T_0 \backslash (T_0^e \cup T_0^{id})))$.

C1b With $schedule(D_0^{id}, Dstate', D_2, D_2^s)$ the rest of the current work is re-scheduled, where:

- $D_2^s = extract2(D_2, \{n | n \in \mathcal{N}_2 \backslash (anc(T_0^e \cup T_0^{id}) \cup T_0^e \cup T_0^{id})\})$.

Fig. 4.9 shows the revised schedule that is in sync with reality again.

4.4.3 New work arrives

At any time, the user can give triggers involving new work, or revision of the current work. Allowed revision of the current work concerns removal or shuffling priorities of orders that are predicted, but not yet in process. Therefore, such revision triggers can be handled by deleting the corresponding prediction and adding new work. In this subsection, the reticle view on another example order (order 2) is used for illustration. This trigger is received at time = 550 [sec], and requests exposure of two wafers according to the 'ABBA' pattern again, but now with reticles RB and RC, respectively.

To react on triggers from the user implying adding new work, six reaction scenarios are defined:

N1a Add new work without current work. The machine is idle when the new work is added. This case is explained in the previous section.

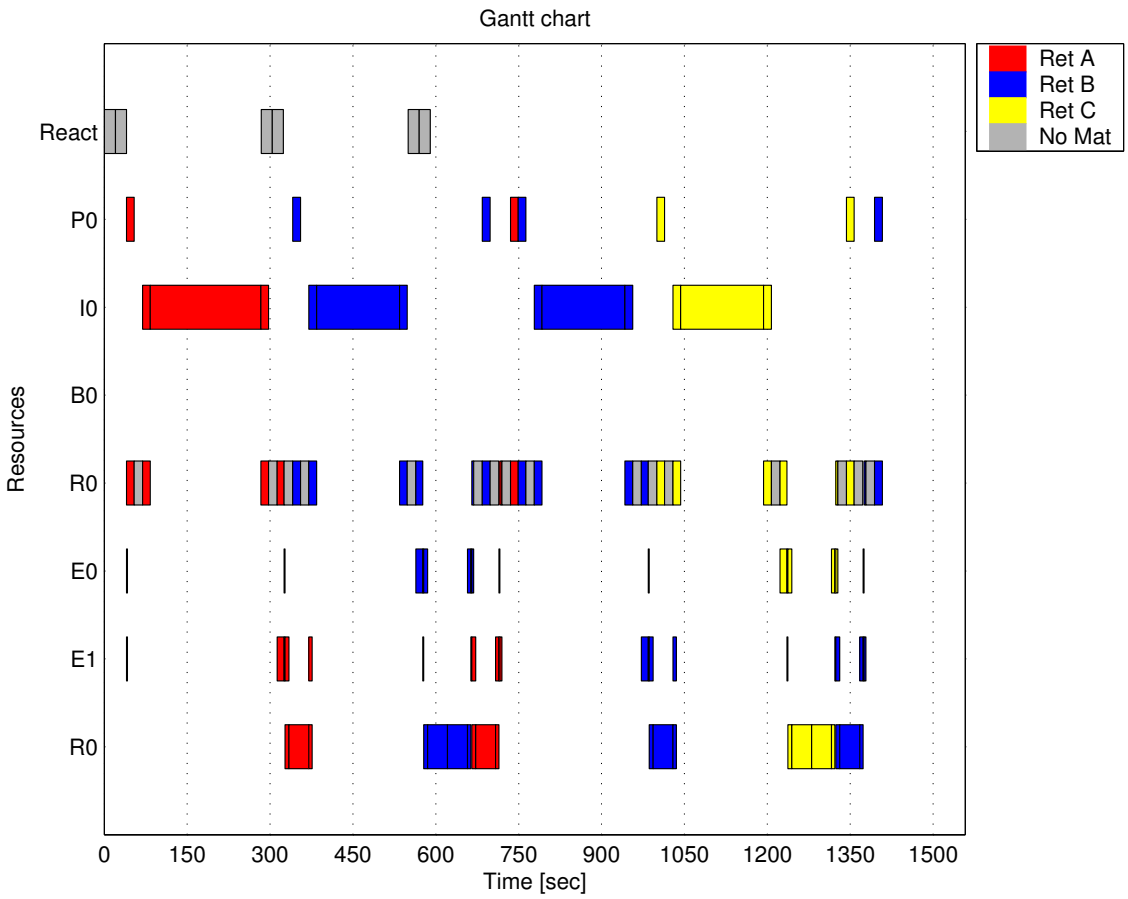


Figure 4.10: Schedule after adding work of order 2

N1b Add new work with current work, no revision of current work. In this case, previously ordered work is still being executed, but its schedule is not revised.

N2a Add new work, re-time current work. In this case, part of the previously ordered work is re-timed such that actual execution is in sync with the schedule again.

N2b Add new work, re-schedule current work. In this case, part of the previously ordered work is re-scheduled together with the new work.

N3a Add new work, re-plan current work. In this case, part of the previously ordered work is re-planned together with the new work.

N3b (potentially) Add new work, re-order current work. In this case, part of the previously ordered work is re-ordered together with the new work. This way, orders can be skipped and their priority can be changed.

Scenario N1a is described in the previous section. Scenario N1b, add new work with current work has two steps:

1. The plan for the new work (order 2), D_2^p, is determined from the state after the current scheduled work $Sstate$ and the order definition D_{2c}^o:

   $D_2^p = plan(Sstate, D_{2c}^o)$.

The resulting precedence graph for order 2 is similar to the precedence graph for order 1 in Fig. 4.5, with wafer W1 and W2 replaced by W3 and W4, and reticle RA and RB replaced by RB and RC, respectively.

2. The schedule for the new work is determined as follows[2]:

   $schedule(\varnothing, Sstate, D_2^p, D_2^p)$.

   In Fig. 4.10, the schedule resulting from this reaction for the example trigger is depicted.

Scenarios N2a and N2b (N2): add new work, re-time or re-schedule current work, are a combination of C1 and N1, involving three steps.

1. First, a scheduling step as described for the C1 scenarios is performed.
2. Besides this, a plan step consisting of two sub steps is done:

   (a) Insert the new order (*no*) after the existing plan, resulting in $D_{2c}^{eids\&no}$
   (b) Plan the result $D_2^{eids\&np} = plan(Fstate, D_{2c}^{eids\&no})$, where *Fstate* is the state after WF.

3. Here, a distinction is to be made between cases N2a and N2b.

   N2a The remainder of the current work is re-timed like in the second step of C1a:
   $(D_0^s, Sstate) = time(D_0^{id}, Dstate, D_2^{eids\&np}, D_1^s)$.
   Subsequently, the new work is scheduled:
   $schedule(D_0^{ids}, Sstate, D_2^{eids\&np}, D_2^{np})$.
   The schedule resulting from this reaction is equal to Fig. 4.10, as re-timing had no effect.

   N2b In case N2b, the remainder of all work is re-scheduled:
   $schedule(D_0^{id}, Dstate, D_2^{eids\&np}, D_2^{s\&np})$.
   In Fig. 4.11, the schedule resulting from this reaction for the example trigger is depicted. This schedule is finished earlier than the schedule in Fig. 4.10, as the two orders are interweaved now during rescheduling.

In the N3 scenarios: add new work, re-plan or re-order current work, besides the work that will not be rescheduled as it remains in the dispatcher (up to WD), there is work that will not be replanned. This is the part of the work scheduled that is generated using the same planning step as work that is in the work up to WD. We call this part of the work scheduled the *initiated* work scheduled. For the example, the status of the nodes in the precedence graph at the arrival of order 2 is depicted including the initiated part of WS, WS-I.

To differentiate the initiated nodes in WS, we define some functions. Let $nodeeid : (\mathcal{N}_{2c}, \mathcal{T}_0^{eid}, \mathcal{D}_{2c}) \rightarrow \mathbb{B}$ be a function that determines whether a node n is an executed, in progress or to be dispatched task, or an ancestor of one, or a parent of nodes that all are:

[2]Assumption: no intra-resource precedence relations between current and new work.

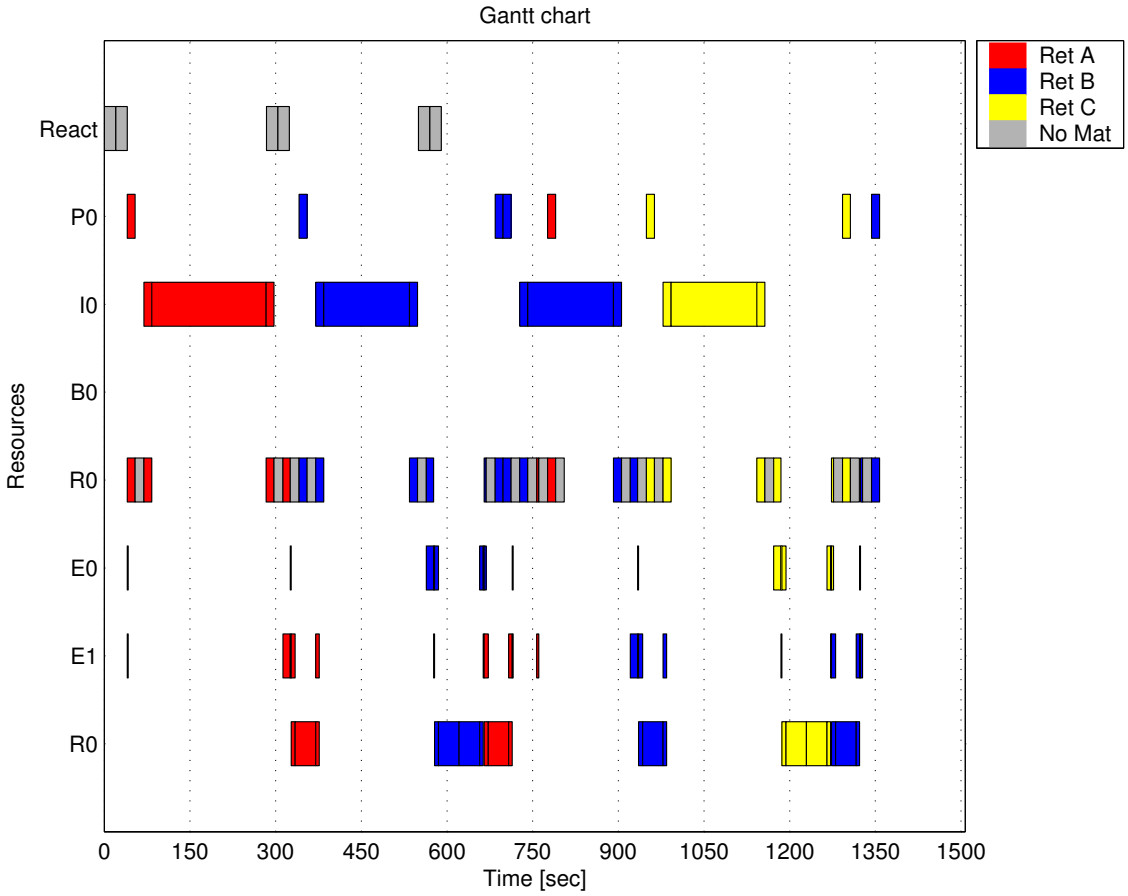


Figure 4.11: Schedule after rescheduling with work of order 2

$$
\begin{aligned}
nodeeid : (n, T_0^{eid}, D_{2c}) = (\ & n \in T_0^{eid} \\
\vee\ & n \in anc(T_0^{eid}) \\
\vee\ & Nr_{2c}(n) \neq \varnothing \wedge (\forall n' : n' \in Nr_{2c}(n) : nodeeid(n')\} \\
)\ &
\end{aligned}
\tag{4.12}
$$

Let $nodeinit : (\mathcal{N}_{2c}, \mathcal{T}_0^{eid}, \mathcal{D}_{2c}) \rightarrow \mathbb{B}$ be a function that determines whether a node n is initiated, defined by:

$nodeinit(n, T_0^{eid}, D_{2c}) =$

$$
\begin{aligned}
& (\exists n' : n' \in Nr_{2c}(n) \cup Ni_{2c}(n) : nodeeid(n', \mathcal{T}_0^{eid}) \vee nodeinit(n', \mathcal{T}_0^{eid})) \\
\vee\ & (\exists n', n'' : n \in Nr_{2c}(n') \cup Ni_{2c}(n'), n'' \in Nr_{2c}(n') \cup Ni_{2c}(n') \backslash \{n\} \\
& : nodeeid(n'', \mathcal{T}_0^{eid}) \vee nodeinit(n'', \mathcal{T}_0^{eid}) \\
&) \\
\vee\ & (\exists n' : n \in Ni_{2c}(n') : nodeeid(n', \mathcal{T}_0^{eid})) \\
\vee\ & (\exists n' : n' \in succ(n) : nodeinit(n', \mathcal{T}_0^{eid}))
\end{aligned}
\tag{4.13}
$$

Then N_{2c}^{eid} can be defined by: $N_{2c}^{eid} = \{n | n \in \mathcal{N}_{2c}, nodeeid(n, T_0^{eid}, D_{2c})\}$, and the initiated nodes, N_{2c}^{init}, can be defined by: $N_{2c}^{init} = \{n | n \in N_{2c} \backslash N_{2c}^{eid}, nodeinit(n, T_0^{eid}, D_{2c})\}$.

Using this, the add new work, re-plan (N3a) or re-order (N3b) current work trigger can be handled in three steps (like N2):

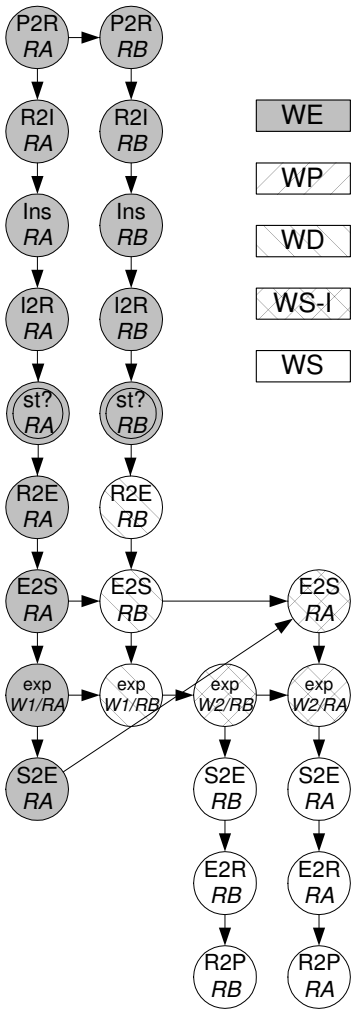


Figure 4.12: Node phase at arrival of order 2

1. A scheduling step as described for the C1 scenarios is performed.
2. Besides this (not necessarily after), a plan of the remainder is derived in four substeps:

   (a) Derive the work up to the initiated work using $extract2c(D_{2c}, N_{2c}^{eid} \cup N_{2c}^{init})$
   (b) Insert after the initiated work the non-initiated order nodes of the current work D_{2c}^{co} (see N2 - step 2a): $N_{2c}^{o} \backslash N_{2c}^{eid} \backslash N_{2c}^{init}$. If required (N3b) in another precedence order.
   (c) Insert after this the new orders D_{2c}^{no} (if any).
   (d) Plan the result: $D_2^{eid\&init\&cp\&np} = plan(Fstate, D_{2c}^{eid\&init\&co\&no})$, where $Fstate$ is the state after WF. This step is visualized in Fig. 4.13.

3. Schedule the result of step 2:

   $schedule(D_0^{id}, Dstate', D_2^{eid\&init\&cp\&np}, D_2^{init\&cp\&np})$.

   In Fig. 4.14, the schedule resulting from this reaction scenario for the example trigger is depicted. This schedule finishes earlier than the schedule in Fig. 4.11, as the post unload and pre load of reticle RB are skipped now during replanning.

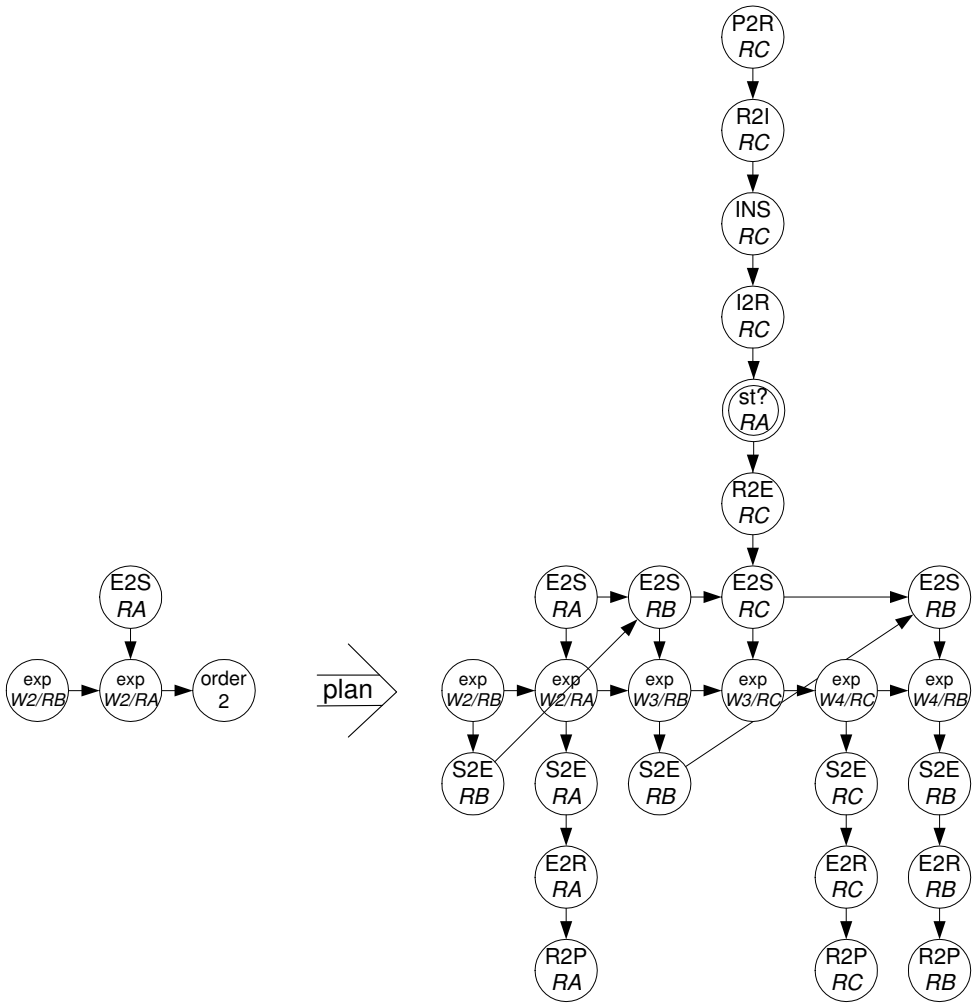


Figure 4.13: Precedence graph before and after replanning with work of order 2

4.4.4 Exceptions occur

Besides the nice weather triggers addressed before, things can also go wrong: exceptions can occur, implying that tasks in the current work fail. Depending on the nature of the failure cause, recovery is or is not possible. Exceptions originating from a defective machine or material in general cannot be recovered. However, many exceptions can be recovered. Although SMC cannot affect the cause of such exceptions, it can affect the effect of them and avoid production loss by automatic recovery. This subsection explains how recovery reaction scenarios can be implemented using the same functionality as reaction to nice weather triggers.

The wafer view on another order (order 3) is used for illustration. This order is received at time = 1050 [sec] and requests to expose six wafers (wafer W5 through W10) with one reticle (RD). The schedule for this order is depicted in Fig. 4.15. The measure task of wafer 7 fails. The status of the nodes at that moment is depicted in Fig. 4.16.

There is a fair chance that this exception can be recovered by re-aligning the wafer. The alignment unit can be reached via the unload robot, using the 'U2A' behavior that is depicted as a dashed arrow in Fig. 4.2. The FIFO (First In, First Out) requirement states that wafers must leave the machine in the same order as the order in which they entered the machine. If the wafers that entered the machine after W7 follow W7 to also

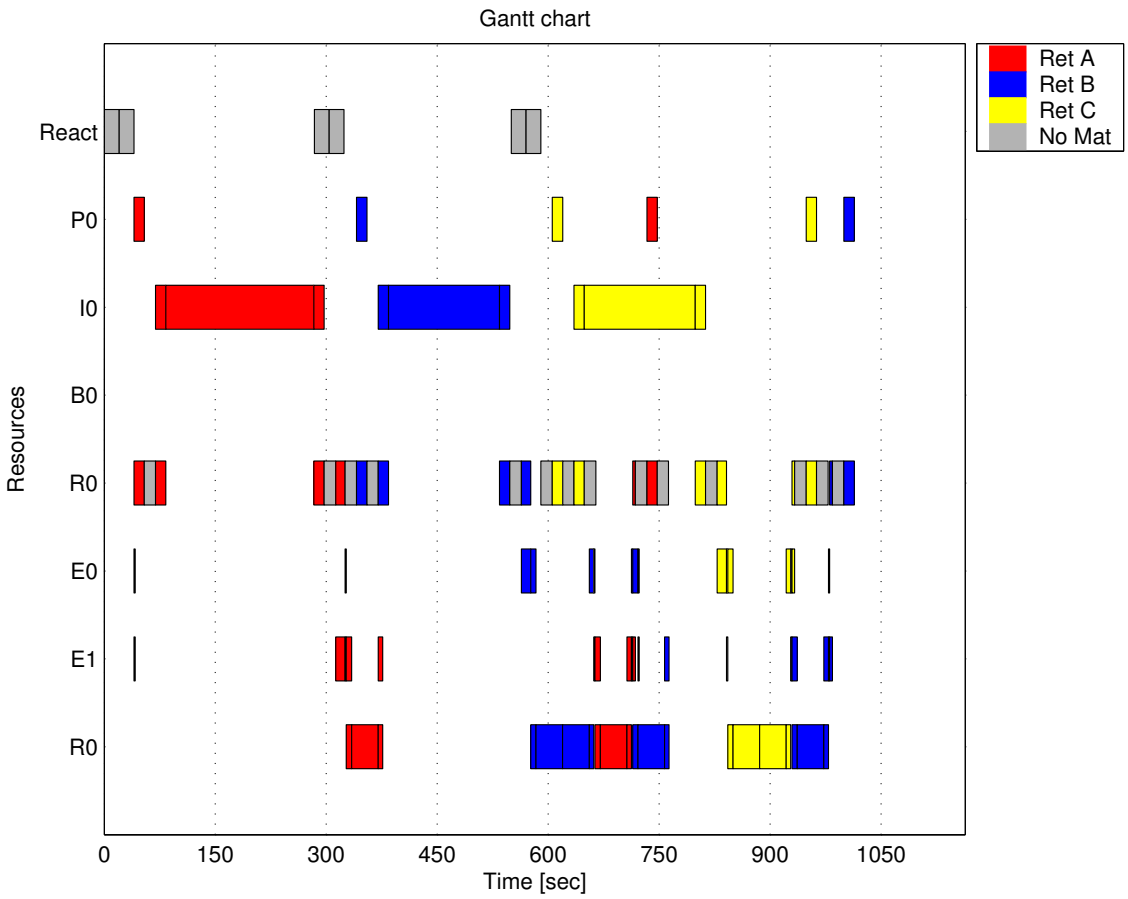


Figure 4.14: Schedule after replanning with work of order 2

make a cycle, this requirement is met.

To recover from an exception by recovery, three steps are to be performed:

1. If the dispatcher receives a trigger that a task has failed, it stops dispatching. The failure message is accompanied by an exception code. The dispatcher waits till the work in process is executed and gathers other exception codes from other tasks that fail, if any.

2. Using the exception information *xinfo*: the set of failed tasks, their exception codes and the system state, the remaining work scheduled (WS) is replanned to process the recovery:

   $D_2^{rec} = recplan(xinfo, D_{2c}^s)$.

   The recovery plan function *recplan* uses a database that maps the exception information onto plan rules and building blocks. In the example case, the plan rules come down to insertion of a recovery building block defining the extra cycle before the remaining lives of wafers W7, W8, and W9. The cycle recovery building blocks can be generated from the cycle building block depicted in Fig. 4.17. Generation comes down to removing the precedence edge that leads to the task that transports the wafer from their current capability to the next capability involved in the cycle, as depicted in Fig. 4.17 for the involved wafers. The generation function that implements this, function *gencrbb*, is shown in the appendix. Furthermore, the measure

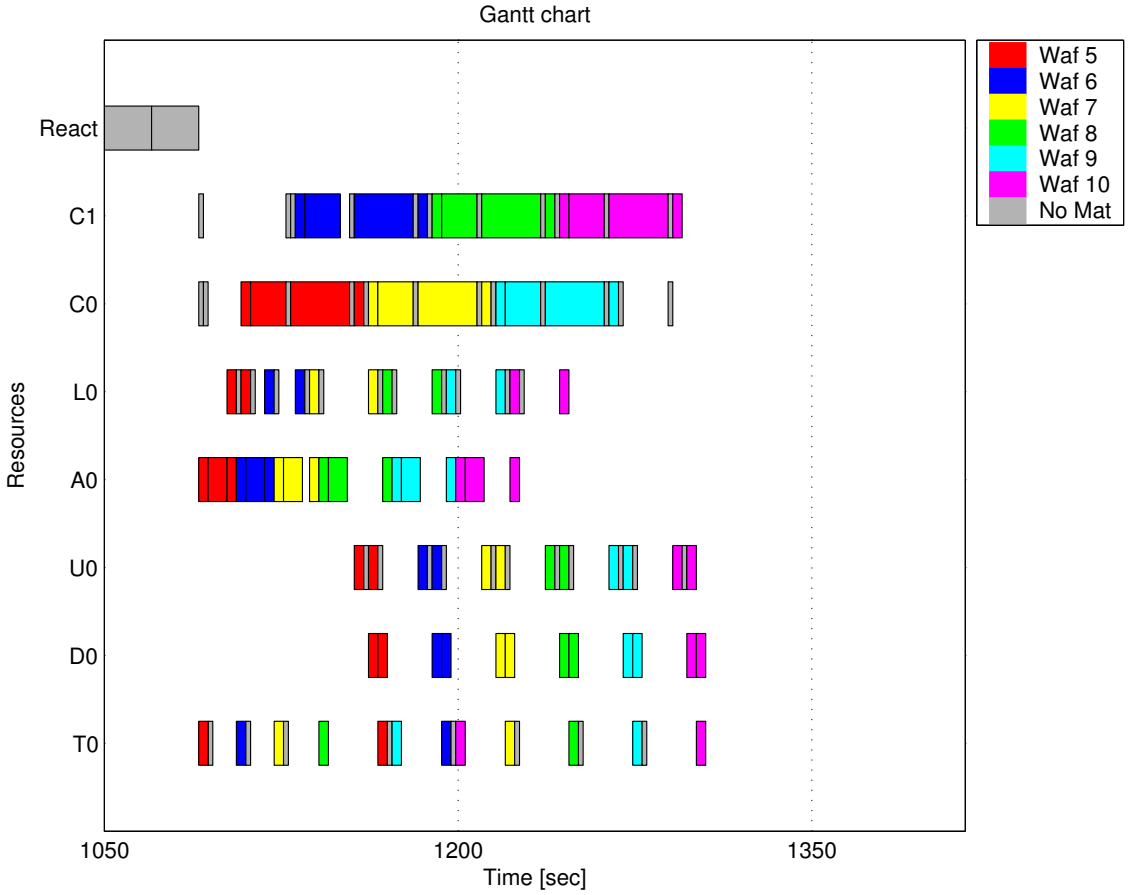


Figure 4.15: Schedule of order 3

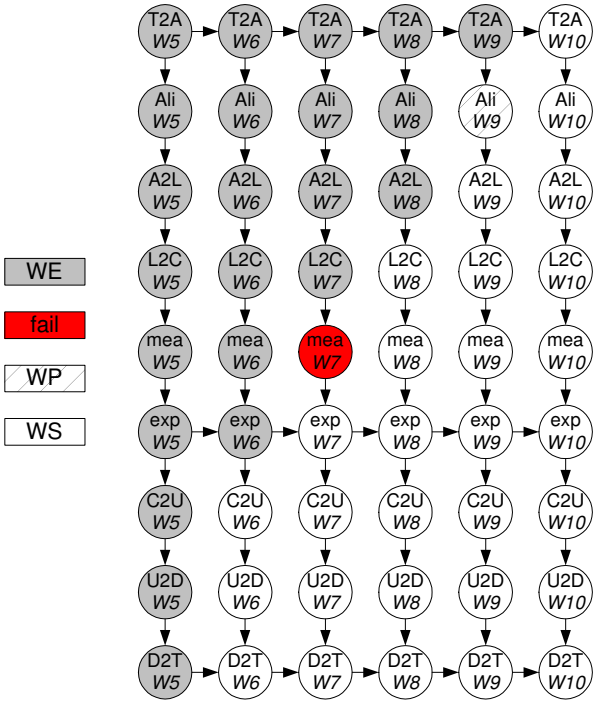


Figure 4.16: Exception: measure of wafer 7 fails

task for W7 must be added[3].

3. There are two possibilities to get a revised schedule including recovery.

   C2a One scenario is to only schedule the recovery part, and to re-time the remainder of the old schedule. However, this scenario cannot be applied straightforwardly in the example case, as the system's material configuration after the recovery part can differ from the system's material configuration before recovery: wafers W7 through W9 can reside at the other wafer chuck.

   C2b A safe scenario is to schedule the entire result of step 2: $schedule(\varnothing, Xstate', D_2^{rec}, D_2^{rec})$, which leads to the schedule depicted in Fig. 4.19.

The revised TRS definition that results after the three insertions is depicted in Fig. 4.18. Also in [10] a database is used to determine schedule revisions, but here job modification is not covered. The final steps of scenarios C1a and C2a can be classified as 'schedule repair', whereas scenarios C1b and C2b can be classified as 'complete regeneration' [22].

4.5 Conclusions

Predictive scheduling is an appropriate approach for scheduling in complex manufacturing machines, as shown in Chapters 2 and 3 ([11, 12]. SMC should be reactive to handle the triggers it receives from its environment. Reactive SMC embedding predictive scheduling requires translation of triggers into scheduling problem instances, which is described in this chapter.

Unlike, for instance, in job shop scheduling, the instantiation of a scheduling problem in a complex manufacturing machine is not straightforward. A rule-based instantiating or planning functionality is explained. Defining planning rules is intuitive as it allows to take the primary manufacturing process as the central axis, and to subsequently wrap the secundary manufacturing processes around this axis. Also drilling down coarse manufacturing steps into steps of a finer grain size is possible.

An SMC framework is described that embeds the layered TRS framework in the form of TRS translation functions. Several methods or scenarios to react to different types of control triggers are described using these translation functions. For the 'nice weather' triggers of type 'current work delays' and 'new work arrives' it is important to avoid control overhead. This is done by making sure that reaction takes place in parrallel with the manufacturing processes if possible. For control triggers involving exceptions it is more important to ensure robust recovery rather than to avoid control overhead. Therefore, reaction to exceptions is sequential. A database provides the necessary information to plan recovery 'patches' in the old scheduling problem instance. Basically, exception recovery uses the same transformation functions as the 'nice weather' triggers mentioned before, which is an elegant characteristic. The framework extends several approaches found in literature [19, 21, 22, 24], which do not consider job construction and modification.

[3]Although this is not the case in the example, a redundant 'Ali' node can result, which must be removed then.

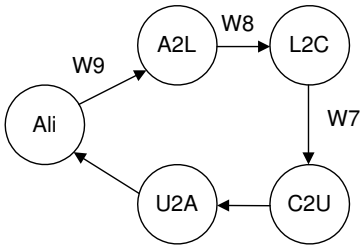


Figure 4.17: Cycle building block

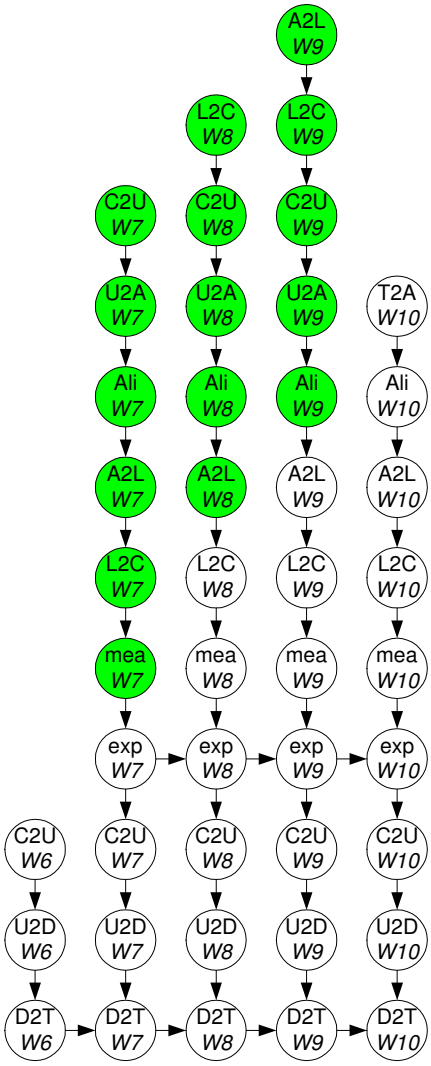


Figure 4.18: Exception recovery: cycle wafers 7 through 9

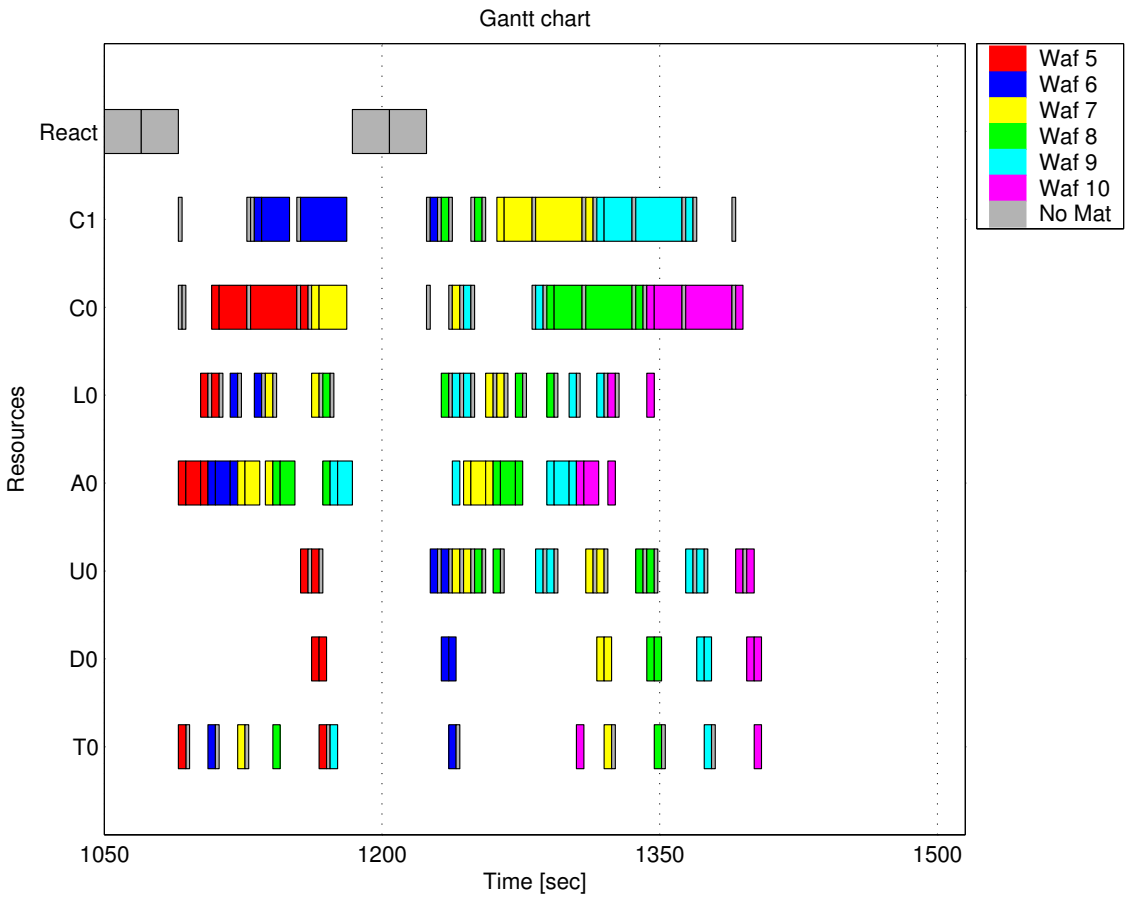


Figure 4.19: Exception recovery: cycle wafers 7 through 9

The SMC framework can be applied in a broad variety of settings. Besides application of one central controller, composition of a hierarchical architecture of controllers is possible. In this case, the tasks of a master controller are the orders for slave controllers (see Fig. 4.7). Note that this hierarchy can also be expressed in the configuration of a single central controller.

The following open issues remain. The combination of all possible things that can go wrong with all material in all possible states at all possible locations implies an exploding number of exceptional states. Although in some exceptions generating functions can be applied to generate the required exception 'patches', it is practically impossible to fill a recovery database for all possible exceptions. Ad-hoc search for exception recoveries is subject of current research.

Acknowledgments

The authors would like to acknowledge Robert Dumont for his help with the case.

Appendices

Generate exposure sequence

For the purpose of the exposure pattern case, the parameters are a tuple of tuples containing a set of capabilities and a list of material instance sets.

Np_{2c}: $\mathcal{N}_2 \rightarrow (\mathcal{P}(\mathcal{C}) \times \mathcal{P}(\mathcal{M})^*)^2$ is an additional element giving the parameters of the nodes.

Let function $addone : \mathcal{D}_{2c} \times \mathcal{D}_{2c} \times (\mathcal{P}(\mathcal{C}) \times \mathcal{P}(\mathcal{M})^*)^2 \rightarrow \mathcal{D}_{2c}$ be a function that adds after an existing system definition D^e_{2c} a basic system definition D^a_{2c}, with materials inherited from np. Function $addone$ can be defined as follows:

$addone(D^e_{2c}, D^a_{2c}, np) = D^{e\prime}_{2c}$ such that: $D^{e\prime}_{2c} = D^e_{2c} \cup D^a_{2c}$ except that:

$$
\begin{aligned}
&(\forall n^{e\prime} : n^{e\prime} \in N^a_{2c} \\
&\quad : (Cb^{e\prime}_{2c}(n^{e\prime}), Ce^{e\prime}_{2c}(n^{e\prime})) = matassign(true, Cb^a_{2c}(n^{e\prime}), Ce^a_{2c}(n^{e\prime}), np) \\
&\quad \wedge ((\nexists n^a : n^a \in N^a_{2c} : n^a \in anc(n^{e\prime}) \vee (n^a, n^{e\prime}) \in P^a_{2c}) \\
&\qquad \Rightarrow \ (\forall n^e : n^e \in N^e_{2c}, (\nexists n : n \in \mathcal{N}^e_{2c} : n \in anc(n^e) \vee (n^e, n) \in P^e_{2c}) \\
&\qquad\qquad : (n^e, n^{e\prime}) \in P^{e\prime}_{2c} \\
&\qquad\quad) \\
&\qquad) \\
&\quad \wedge Np^{e\prime}_{2c}(n^{e\prime}) = ((np.0.0, [hd(np.0.1)]), (np.1.0, [hd(np.1.1)])) \\
&)
\end{aligned}
\tag{4.14}
$$

Using this, function $genes : \mathcal{D}_{2c} \times \mathbb{B} \times (\mathcal{P}(\mathcal{C}) \times \mathcal{P}(\mathcal{M})^*)^2 \times \mathbb{N} \times \mathbb{N} \times \mathcal{D}_{2c} \rightarrow \mathcal{D}_{2c}$ can be defined recursively. The $genes$ function is suited to generate an exposure sequence for a wide range of exposure patterns, for processing wafers in pairs and for any number of reticles: e.g. patterns A, ABBA, ABAB, ABCABC, AABB, AABBCCDDDDCCBBAA etc.

$$
\begin{aligned}
&genes(D^a_{2c}, prs, np, RpP, WpP, D^e_{2c}) = \\
&\begin{cases}
D^e_{2c} & \text{if } len(np.0.1) = 0 \\
genes(D^a_{2c}, prs, np', RpP', WpP', addone(D^e_{2c}, D^a_{2c}, np)) & \text{if } len(np.0.1) > 0
\end{cases}
\end{aligned}
\tag{4.15}
$$

Where:

- D^a_{2c} is the basic system definition used for generation, e.g. an exposure task including all involved elements.
- prs defines whether wafers are processed per pair or not, e.g. false for 'ABBA' (DET) and true for 'AABB'.
- np are the node parameters, e.g. (({WS, PA, ...}, [{W1}, {W2}, {W3}, {W4}, {W5}]), ({RS, Turret, ...}, [{R1}, {R2},{R2}, {R1}])) for a five-wafer lot with wafers W1 through W5 that must be exposed following the 'ABBA' pattern with reticles R1 and R2.
- RpP is the number of reticles to go from that point to finish this pair of wafers (or wafer).
- WpP is the number of wafers to go from that point to finish this pair of wafers for this reticle.
- D^e_{2c} is the addition generated up to that point.

In Table 4.1, the variables used in the definition above are summarized.

Table 4.1: Definition of variables used in function *genes*.

prs	$len(np.0.1)$	RpP	WpP	np'	RpP'	WpP'
$false$	> 0	1	1	$($ $(np.0.0, np'_3)$ $,$ $(np.1.0, np'_1)$ $)$	$size(el(np.1.1))$	1
$false$	> 0	> 1	1	$($ $np.0$ $,$ $(np.1.0, np'_1)$ $)$	$RpP - 1$	1
$true$	1	1	1	$($ $(np.0.0, np'_3)$ $,$ $(np.1.0$ $, tl(tl(np.1.1)))$ $)$	RpP	1
$true$	1	> 1	1	$($ $np.0$ $,$ $(np.1.0$ $, tl(tl(np.1.1)))$ $)$	$RpP - 1$	1
$true$	> 1	> 1	2	$($ $(np.0.0, np'_2)$ $,$ $(np.1.0, np'_1)$ $)$	RpP	1
$true$	> 1	> 1	1	$($ $(np.0.0, np'_2)$ $,$ $(np.1.0, np'_1)$ $)$	1	2
$true$	> 1	1	2	$($ $(np.0.0, np'_3)$ $,$ $(np.1.0, np'_1)$ $)$	RpP	1
$true$	> 1	1	1	$($ $(np.0.0, np'_3)$ $,$ $(np.1.0, np'_1)$ $)$	$size(el(np.1.1))$	2

where:
$np'_1 = tl(np.1.1) +\!\!+ [hd(np.1.1)]$
$np'_2 = [hd(tl(np.0.1))] +\!\!+ [hd(np.0.1)] +\!\!+ tl(tl(np.0.1))$
$np'_3 = tl(np.0.1)$

Generate cycle recovery building blocks

Function $gencrbb : \mathcal{D}_{2c} \times \mathcal{M} \times \mathcal{C} \rightarrow \mathcal{D}_{2c}$ instantiates for material m residing at capability ml a cycle recovery building block $D^{a\prime}_{2c}$ from the cycle building block D^{a}_{2c} depicted in Fig. 4.17. Such a generated building block can replace a general recovery node that is inserted before the remainder of the plan.

$gencrbb(D^{a}_{2c}, m, ml) = D^{a\prime}_{2c}$ such that: $D^{a\prime}_{2c} = D^{a}_{2c}$ except that:

$$
\begin{array}{lll}
 & (\forall n, n' & : (n, n') \in P^{a}_{2c}, Cb^{a}_{2c}(n', ml) \neq \varnothing, Ce^{a}_{2c}(n', ml) = \varnothing \\
 & & : (n, n') \notin P^{a}_{2c} \\
 &) & \\
\wedge & (\forall n, c & : n \in N^{a}_{2c}, c \in I^{a}_{2c}(n) \\
 & & : (Cb^{a}_{2c}(n, c) \neq \varnothing \Rightarrow Cb^{a\prime}_{2c}(n, c) = \{m\}) \\
 & & \wedge (Ce^{a}_{2c}(n, c) \neq \varnothing \Rightarrow Ce^{a\prime}_{2c}(n, c) = \{m\}) \\
 &) &
\end{array}
\tag{4.16}
$$

References

[1] R. J. Abumaizar and J. A. Svestka. Rescheduling job shops under random disruptions. *International journal of production research*, 35(7):2065–2082, 1997.

[2] ASML, 2004. Information on wafer scanners available through URL `http://www.asml.com/`, item: products - lithography.

[3] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–341, 1994.

[4] H. Chen and B. Hu. Schedule-driven supervisory control of flexible manufacturing systems. In *30th Conference on Decision and Control*, pages 2186–2191, 1991.

[5] S. Chung, S. Lafortune, and F. Lin. Limited lookahead policies in supervisory control of discrete event systems. *IEEE Transactions on Automatic Control*, 37(12):1921–1935, 1992.

[6] J. Dorn, R. Kerr, and G. Thalhammer. Reactive scheduling. *International journal of human-computer studies*, 42:687–704, 1995.

[7] M. P. Fanti and M. Zhou. Deadlock control methods in automated manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 34(1):5–22, 2004.

[8] P. Gohari and W. M. Wonham. Reduced supervisors for timed discrete-event systems. *IEEE Transactions on Automatic Control*, 48(7):1187–1198, 2003.

[9] Y. Li and Z. H. Lin. Supervisory control of probabilistic discrete-event systems with recovery. *IEEE Transactions on Automatic Control*, 44(10):1971–1975, 1999.

[10] K. Miyashita and K. Sycara. CABINS: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair. *Artificial Intelligence Journal*, 76(1-2):377–426, 1995.

[11] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, N. C. W. M. Braspenning, and J. E. Rooda. Predictive scheduling in complex manufacturing machines: scheduling alternatives and algorithm. *submitted to IEEE TAC*.

[12] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, N. C. W. M. Braspenning, and J. E. Rooda. Predictive scheduling in complex manufacturing machines: machine-specific constraints. *submitted to IEEE TSM*.

[13] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, and J. E. Rooda. Design of supervisory machine control. In K. Glover and J. Maciejowski, editors, *Proceedings of the European Control Conference 2003*, 2003. CD-ROM.

[14] D. Ouelhadj, P. I. Cowling, and S. Petrovic. Utility and stability measures for agent-based dynamic scheduling of steel continuous casting. In *IEEE International Conference on Robotics & Automation*, pages 175–180, 2003.

[15] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 1995.

[16] R. G. Qiu and S. B. Joshi. A structured adaptive supervisory control methodology for modeling the control of a discrete event manufacturing system. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 29(6):573–586, 1999.

[17] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[18] A. Ramirez-Serrano and B. Benhabib. Supervisory control of flexible-manufacturing workcells that allow the production of a priori unplanned part types. In *IEEE International Conference of Systems, Man and Cybernetics*, pages 2127–2131, 2000.

[19] I. Sabuncuoglu and M. Bayiz. Analysis of reactive scheduling problems in a job shop environment. *European journal of operational research*, 126:567–586, 2000.

[20] S. F. Smith. Is scheduling a solved problem? In G. Kendall, E. Burke, and S. Petrovic, editors, *Multidisciplinary International Conference on Scheduling : Theory and Applications(MISTA'03)*, pages 11–20. ASAP, University of Nottingham, UK, August 2003.

[21] E. Szelke and R. M. Kerr. Knowledge-based reactive scheduling. In *Proceedings of the IFIP TC5/WG5.7 international workshop*, 1993.

[22] G. E. Vieira, J. W. Herrmann, and E. Lin. Rescheduling manufacturing systems: a framework of strategies, policies and methods. *Journal of scheduling*, 6(1):35–58, 2003.

[23] H. H. Wu and R. K. Li. A new rescheduling method for computer based scheduling systems. *International journal of production research*, 33(8):2097–2110, 1995.

[24] M. Zweben and M. S. Fox. *Intelligent scheduling*. San Francisco: Morgan Kaufmann, 1994.

CHAPTER

FIVE

EXCEPTION RECOVERY SEARCH IN COMPLEX MANUFACTURING MACHINES

This chapter contains the paper *Exception Recovery Search in Complex Manufacturing Machines* that has been protected in patent application ASML ref. P-1704. The idea has been filed with the first filing in Europe in October 2003, number 03256456.9. The paper text is in the subsequent filing of September and October 2004, which is also filed in Japan (number 2004-286595) and the US.

Exception recovery search in complex manufacturing machines

N.J.M. van den Nieuwelaar †*, J.M. van de Mortel-Fronczak †,
R.J. Dumont †, J.E. Rooda †

Abstract

As the number of exceptional states in a complex manufacturing machine is very large and buffer space is limited, it is practically impossible to predefine all exception recoveries. In this chapter, an approach is presented that can be embedded in Supervisory Machine Control (SMC) to ad-hoc, run-time search for an exception recovery once an exception is encountered. The manufacturing possibilities of the machine are specified by so-called meta-tasks, having pre-conditions and post-conditions. Pre-conditions define in which states of the machine the meta-task can successfully be executed, whereas post-conditions define the expected machine state after execution of the meta-task. A search algorithm is proposed that starts from an exceptional state and searches for an exception recovery reaching a desired state using the meta-tasks. The instantiated recovery is formulated in the form of a scheduling problem that can be scheduled and dispatched afterwards using existing theory. The approach is illustrated using a representative example of a complex manufacturing machine: a wafer scanner.

5.1 Introduction

A complex manufacturing machine consists of many mechatronic systems and can process many different types of products. Supervisory Machine Control (SMC) co-ordinates these mechatronic systems: it decides when to do which manufacturing tasks using which mechatronic subsystem resources. Many options exist to deploy the available resources to perform tasks that lead to the desired manufacturing purpose, resulting in various machine behaviors. SMC should optimize machine behavior, and what is best may depend on the characteristics of the product recipe. Furthermore, SMC should properly react to all kinds of triggers from the environment. A very important trigger is task failure: an exception, which requires a recovery reaction of SMC to avoid human intervention.

5.1.1 Literature

Many approaches exist to describe a system under supervisory control using well-known formalisms from computer science. Supervisory control theory (SCT) as discussed by Wonham et al. [2, 7, 14] models the system under control using Finite State Machines. The possible behavior of such a system is regarded as a language. A supervisory controller in the form of a deterministic automaton is synthesized that restricts the language by disabling a subset of events, to control the system to properly accomplish its task.

† Eindhoven University of Technology: P.O. box 513, 5600 MB Eindhoven, The Netherlands.
* ASML: De Run 6501, 5504 DR Veldhoven, The Netherlands.
Corresponding author: N.J.M. van den Nieuwelaar, e-mail: n.j.m.v.d.nieuwelaar@tue.nl

Exception recovery in manufacturing systems described in literature is restricted to local recovery. A task is retried at the same resource [8], at another equivalent resource [13], or recovery is restricted to one product [3, 4, 15]. However, in complex manufacturing machines it can be the case that in order to recover a previous manufacturing step at another location in the machine must be executed again. Moreover, getting the product over there might involve moving other products, as buffer places are limited (in contrast to what is assumed in [3, 4, 15]).

A predictive-reactive scheduling approach is proposed in Chapter 4 that is also suited for non-local exception recovery in complex manufacturing machines. It is based on the predictive scheduling approach presented in Chapters 2 and 3 ([9, 10]) and is therefore better suited for optimization than SCT-based approaches due to the expressivity for scheduling alternatives and the use of predictive information. To react to exceptions a database of predefined recoveries is used.

However, it is practically impossible to define how to recover for all possible exceptions during design. This is caused by the fact that combination of all possible things that can go wrong with all material instances in all possible states at all possible locations implies an exploding number of exceptional states. However, the number of different manufacturing steps that a machine can perform is not so large. Furthermore, the amount of computing power on board in complex manufacturing machines is huge. Therefore, this chapter proposes ad-hoc, run-time search for exception recovery from an encountered exceptional state.

To search for a path from one location to another one is well-studied in route planning literature [5]. However, also the other products in the machine must be taken into account. They can block the path of the product to be recovered, and this blocking can be resolved by SMC itself. In that sense, the recovery search problem resembles a board game, in which the location of the pieces also determines which moves are possible [6]. In complex manufacturing machines not only the moves play a role, but also the manufacturing process must be taken into account.

5.1.2 Layered task resource system framework

From the SMC point of view, a machine can be considered as a task resource system (TRS). Tasks can be associated with manufacturing processes, whereas resources can be associated with mechatronic systems. Transforming a manufacturing request into machine behavior can be structured in three phases. First, a scheduling problem must be instantiated from the manufacturing request, taking into account the limitations of the machine. This transformation is called *instantiating*. The structure of the resulting scheduling problem shows many similarities with the job shop scheduling problem [12]. The manufacturing process of a material instance can be associated with a job, whereas the different parallel mechatronic systems can be associated with the different machines in a job shop. Subsequently, resources must be assigned to the tasks in the instantiated scheduling problem in some order, taking into account the fact that resources are able to perform certain tasks only, and only one at a time. This transformation is called *selecting*. The selected order of tasks to be performed by selected resources implies consecutive state transitions of those resources, which is analogous to the setup times for mode switching in job shop scheduling. Finally, start and finish times can be assigned to the tasks, taking into account the speed of the resources. This transformation is called *timing*.

During the three transformation phases of instantiating, selecting and timing, choices

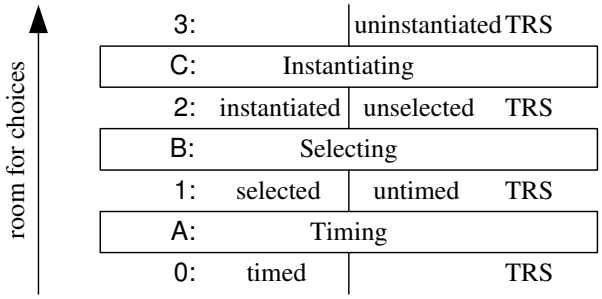


Figure 5.1: Layered Task Resource System framework

must be made. The result of a choice in a certain transformation on the machine behavior can only be evaluated by performing the consecutive transformations. Therefore, a transformation phase strongly relies on information from subsequent phases. The layered TRS framework shown in Fig. 5.1 displays the hierarchically related transformation phases as functionality layers (A through C) and the different TRS definition levels (0 through 3) as interfaces between the layers (see Chapter 1 and [11]).

5.1.3 Structure of the chapter

The structure of this chapter is as follows. Throughout the chapter, an example of a complex machine is used for illustration: a dual-stage wafer scanner [1]. Section 5.2 describes how wafers are processed in such a wafer scanner. Besides nice weather operation, also the manufacturing steps that can be useful to recover from exceptions are explained. Section 5.3 defines the starting point of the exception recovery search: the system state and the uninstantiated TRS definition (see Fig. 5.1). Moreover, capturing of the manufacturing steps of the wafer scanner example in such a TRS definition is demonstrated. In Section 5.4, the instantiating functionality that transforms an uninstantiated TRS definition into an instantiated unselected TRS definition (scheduling problem) is explained. In case of exception recovery, the instantiating functionality searches for a way to reach a recovered target state from the exceptional start state, resulting in a recovery in the form of a scheduling problem. This section addresses the instantiating constraints, state updating, and a search algorithm that is suited for practical use. Section 5.5 shows the result of application of the approach for an example exception in the wafer scanner example. Finally, concluding remarks are presented in Section 5.6.

5.2 Wafer processing in a wafer scanner

In this section, first wafer processing in a wafer scanner is explained. After that, the result of instantiating is defined: an instantiated, unselected TRS or scheduling problem. Finally, a scheduling model and a schedule are illustrated using an example order.

5.2.1 Wafer scanner description

The primary manufacturing process of a wafer scanner is the exposure of a mask containing an IC pattern onto wafers. Fig. 5.2 shows a schematic layout of a dual-stage wafer

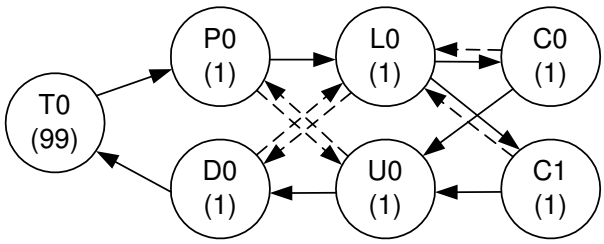


Figure 5.2: Layout

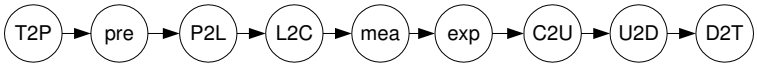


Figure 5.3: Nice weather life of a wafer

scanner. In this figure, circles depict the mechatronic subsystems or resources regarded here, and arrows depict the possible transport paths. The number of material instances (wafers) that a resource can carry is depicted between brackets. At the right side of the figure, the two wafer chucks that are present in a dual-stage wafer scanner (C0 and C1) are depicted, that carry the wafer during exposure. As the required accuracy of the exposure process is very high, any imperfections concerning the wafers must be corrected for. To be able to do this, wafers are measured at a wafer chuck before being exposed. There are separate measure and expose areas in the machine. The orientation of the wafer at a wafer stage is of importance for successful measurement and exposure, whereas the orientation is unknown when a wafer comes into the machine. Therefore, a pre-alignment unit (P0) is incorporated. A neighboring machine named track (T0) performs some pre-processing and post-processing steps, and delivers wafers to the alignment system. A load robot (L0) transports wafers between the pre-alignment system, the wafer chucks and the discharge unit. An unload robot (U0) can do the same, but cannot transport wafers to the wafer chucks. Loading onto and from a chuck is only possible if it is at the measure area of the machine. From the discharge unit (D0), wafers are picked up by the track.

The steps in the manufacturing process of a wafer ('life of a wafer') are as follows. First, the wafer is transported from the track onto the pre-alignment unit (T2P). Subsequently, the pre-alignment takes place (pre). After that, the load robot takes the wafer from the alignment unit (P2L), and places the wafer onto a chuck (L2C). On the chuck, the wafer is measured (mea) and, subsequently, exposed (exp). Then, the unload robot takes the wafer from the chuck (C2U) and puts the wafer onto the discharge unit (U2D). Finally, the wafer is taken from the discharge unit by the track (D2T). A precedence graph of the life of a wafer is depicted in Fig. 5.3.

Two types of manufacturing steps can be distinguished in the life of a wafer: process steps and transport steps. Process steps take place at one location, whereas transport steps move a wafer from one location to another and are depicted as arrows in Fig. 5.2. However, Fig. 5.2 contains also dashed arrows. These arrows concern transportation possibilities that are not involved in the nice weather life of a wafer, but can be useful for recovery purposes. In any case, it is important that wafers leave the machine in the same order as they entered it (FIFO).

5.2.2 Scheduling model definition

In Chapter 3 ([10]), the scheduling model of a complex machine with its physical restrictions is defined. For the purpose of this chapter, the elements needed for the timing transformation (see Fig. 5.1) are not relevant. Without them, the scheduling model can be defined by a 12–tuple:
$(\mathcal{T}_2, \mathcal{R}, \mathcal{C}, I_2, A, P_2, Pt_2, \mathcal{M}, Cb_2, Ce_2, Rm, Mf\ )$:

- $\mathcal{T}_2$ is a finite set of elements called tasks.
- $\mathcal{R}$ is a finite set of elements called resources.
- $\mathcal{C}$ is a finite set of elements called capabilities.
- I_2: $\mathcal{T}_2 \rightarrow \mathcal{P}(\mathcal{C})$ gives the set of capabilities that are involved in a certain task.
- A: $\mathcal{C} \rightarrow \mathcal{P}(\mathcal{R})$ gives the set of resources that are available for a certain capability.
- $P_2 \subseteq \mathcal{T}_2 \times \mathcal{T}_2$ is the precedence relation between tasks.
- $Pt_2 \subseteq \mathcal{T}_2 \times \mathcal{T}_2$ is the tied precedence relation between tasks.
- $\mathcal{M}$ is a finite set whose elements are called material instances.
- Cb_2, Ce_2: $TC \rightarrow \mathcal{P}(\mathcal{M})$ give the begin and the end material configuration of each capability involved in a certain task, where $TC = \{(t, c) | t \in \mathcal{T}_2, c \in I_2(t)\}$
- Rm: $\mathcal{R} \rightarrow \mathbb{N}$ gives the number of material instances that can reside on a certain resource.
- $Mf \subseteq \mathcal{R} \rightarrow \mathcal{R}$ represents the physically possible material flow as a set of tuples defining from which resource to which resource material can flow.

5.2.3 An instantiated scheduling model and schedule

The scheduling model in a complex machine can be split into two sections: system-dependent elements and work-dependent elements. The system-dependent elements can be instantiated using Fig. 5.2 as follows:

- There are six capabilities: the chuck, the load and unload robots, the alignment and discharge units, and the track.
  $\mathcal{C}$ = {C, L, U, P, D, T}.
- There are seven resources: one for each capability, except two wafer chucks:
  $\mathcal{R}$ = {C0, C1, L0, U0, P0, D0, T0}.
- The available resources for each capability are defined as follows:
  A = {(C, {C0, C1}), (L, {L0}), (U, {U0}), (P, {P0}), (D, {D0}), (T, {T0})}.
- The material capacity of the resources is one for each resource, except for the track, the pod, and the buffer:

  Rm ={(C0, 1), (C1,1), (L0,1), (U0,1), (P0,1), (D0,1), (T0,99)}.
- The possible material flow is defined as follows:

  Mf = {(T0, P0), (P0, L0), (L0, C0), (L0, C1), ...}.

As an example, a batch or lot of six wafers, wafer *W1* through *W6*, is processed. The precedence graph for this lot is depicted in Fig. 5.4. It consists of six wafer lives, of which the first and the 'exp' steps are connected with precedence edges to ensure FIFO processing. The work-dependent elements for such a lot can be instantiated as follows:

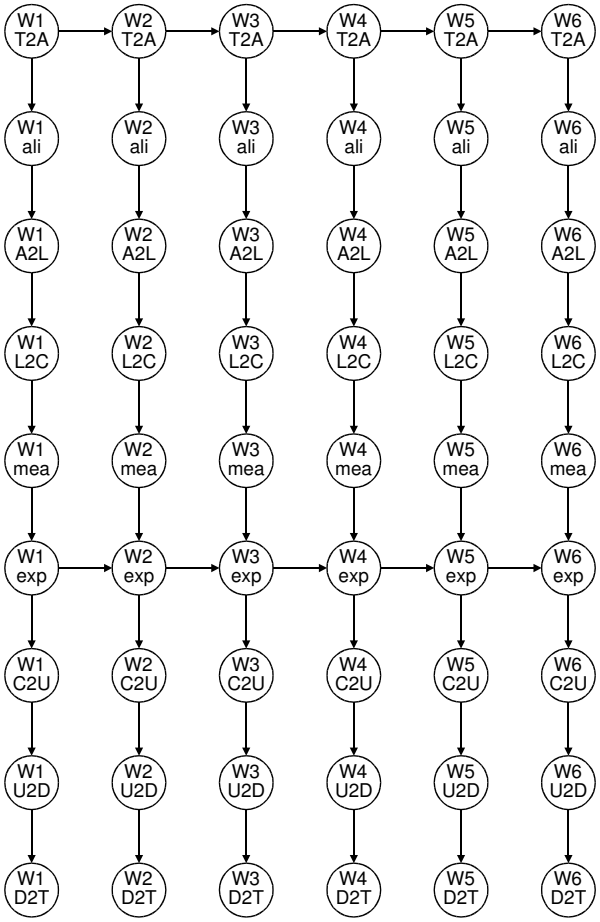


Figure 5.4: Initial precedence graph

- $\mathcal{T}_2 = \{$W1-T2P, W1-pre, ..., W2-T2P, W2-pre, ...$\}$.
- $I_2 = \{$(W1-T2P, {T, P}), (W1-pre, {P}), ...$\}$.
- $P_2 = \{$(W1-T2P, W1-pre), (W1-pre, W1-P2L), ..., (W1-T2P, W2-T2P), (W2-T2P, W2-pre), ...$\}$.
- $Pt_2 = \{\}$.
- $\mathcal{M} = \{$W1, W2, W3, W4, W5, W6$\}$.
- $Cb_2 = \{$((W1-T2P, T), {W1}), ((W1-T2P, P), {}), ((W1-pre, P), {W1}), ...$\}$.
- $Ce_2 = \{$((W1-T2P, T), {}), ((W1-T2P, P), {W1}), ...$\}$.

A schedule for this lot is shown in Fig. 5.5.

5.3 Uninstantiated system definition

Whereas tasks in an instantiated TRS (see Fig. 5.1) have material instances assigned to them and a precedence relation, this is not the case for an uninstantiated TRS. Tasks can be instantiated from so-called meta-tasks. Meta-tasks can be associated with the possible manufacturing steps described in the previous section. Although there is no precedence relation between meta-tasks, constraints exist for instantiating meta-tasks, ensuring that the resulting instantiated TRS definition is feasible. Instantiation of a transport step or

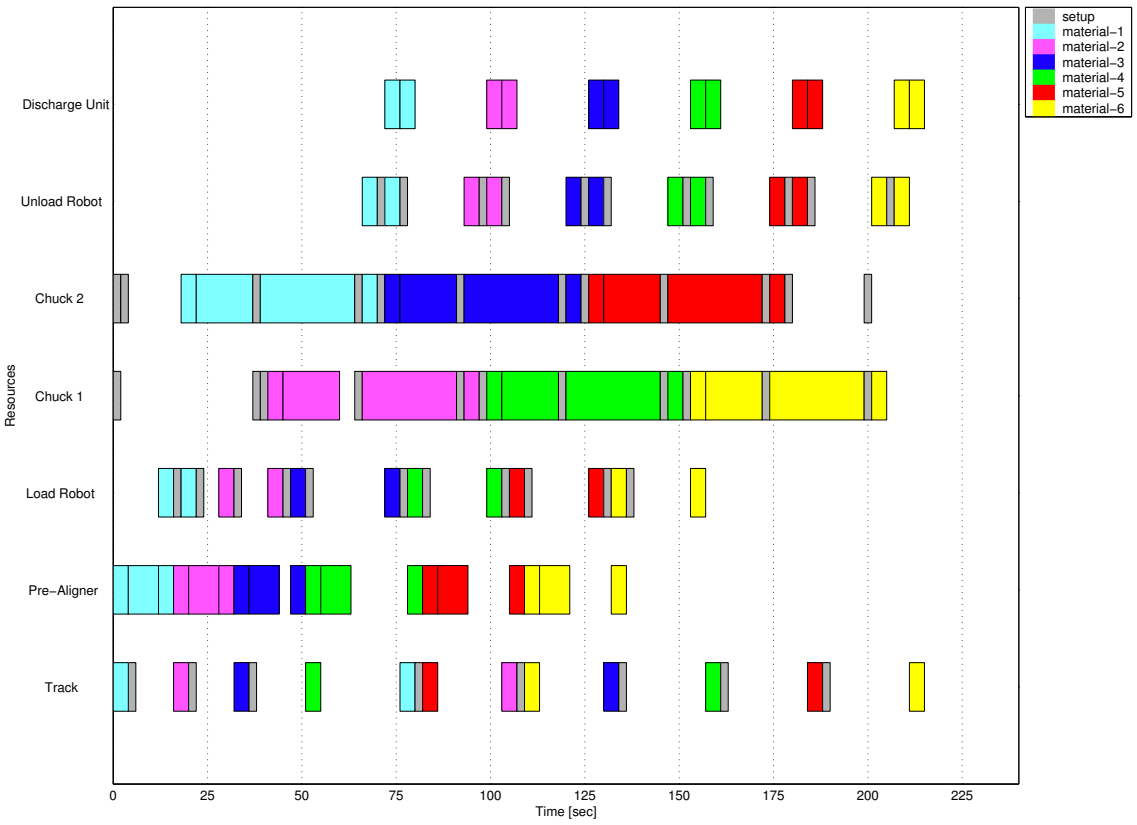


Figure 5.5: Initial schedule

meta-task is only possible if there is material available at the 'from' location, and there is enough room at the 'to' location. For example: picking up a wafer from the alignment unit by a robot is only possible if there is a wafer at the alignment unit, and the robot has no wafer yet. Furthermore, a process can only succeed if a wafer is at the right location in the right state. For example: measuring an aligned wafer at a wafer chuck.

For the purpose of this chapter, the following is assumed:

1. Resources r, r' of the same capability c are 'equivalent': they have the same material capacity (Rm) and logistic connections (Mf). Furthermore, transportation of material between resources of the same capability is not possible. These constraints can be defined as follows.

$$\begin{aligned}(\forall c, r, r' \quad & : c \in \mathcal{C}, r, r' \in A(c) \\ & : Rm(r) = Rm(r') \\ & \wedge (\forall r'' : (r'', r) \in Mf : (r'', r') \in Mf) \\ & \wedge (\forall r'' : (r, r'') \in Mf : (r', r'') \in Mf) \\ & \wedge (r, r') \notin Mf \wedge (r', r) \notin Mf \\ )\end{aligned} \tag{5.1}$$

As a consequence, a capability can be regarded as a material location.

2. The manufacturing phases of a material instance can be expressed using linear, unit-distant material manufacturing phase identifiers $\mathcal{S}_m \subseteq \mathbb{N}$. These identifiers and their description are as follows for the wafers in the example:

   - 0: fresh
   - 1: pre-aligned
   - 2: measured
   - 3: exposed

Under these assumptions, the system state can be defined by $Ssys : \mathcal{C} \rightarrow \mathcal{P}(\mathcal{M} \times \mathcal{S}_m)$, describing for each capability which material instances reside on it and their manufacturing phase. Based on this system state, the definition elements concerning the meta-tasks of the uninstantiated TRS definition can be formulated:

- $\mathcal{T}_3$ is a finite set of uninstantiated tasks or meta-tasks.
- I_3: $\mathcal{T}_3 \rightarrow \mathcal{P}(\mathcal{C})$ gives the set of capabilities that are involved in a certain meta-task.
- Cb_3, Ce_3: $TC \rightarrow \mathcal{P}(\mathbb{N} \times \mathcal{S}_m)$ give the pre-condition at the beginning and expected post-condition at the end of a meta-task, where $TC = \{(t, c) | t \in \mathcal{T}_3, c \in I_3(t)\}$. The conditions are defined by a tuple per involved capability: the number of material instances involved and their manufacturing phase.

In case of a process meta-task, the number of material instances at the beginning and their manufacturing phase form the pre-condition for instantiating. The end manufacturing phase defines the manufacturing phase of a material instance resulting from the meta-task, e.g. the pre-alignment meta-task 'pre': Cb_3('pre', 'P') = (1,0), Ce_3('pre', 'P') = (1, 1).

In case of a transport meta-task, only the numbers of material instances form the instantiating pre-condition. The manufacturing phase of material after transport depends on the manufacturing phase before transport. It remains unchanged if the manufacturing phase was equal to the required manufacturing phase at the beginning (pre-condition), and is reset to the end manufacturing phase (post-condition) if not. For example the conditions of meta-task 'C2U' describe that a wafer is transported from a chuck to the unload robot, and keeps its manufacturing phase only if this was phase *3* ('exposed'): Cb_3('C2U', 'C') = (1, *3*), Cb_3('C2U', 'U') = (0, 0), and that it is reset to *0* if not: Ce_3('C2U', 'C') = (0, 0), Ce_3('C2U', 'U') = (1, *0*). So, e.g. a 'measured' wafer (phase < 3) becomes 'fresh' (phase 0) when it is transported from a chuck to the unload robot. Table 5.1 gives an overview of all pre- and post-conditions of the meta-tasks in the example.

Given the definition elements defining the meta-tasks, the types of meta-tasks can be derived using function

Table 5.1: Pre-conditions and expected post-conditions.

meta-task , capability (t, c)	pre-condition $Cb_3(t, c)$	post-condition $Ce_3(t, c)$
(pre, P)	(1, 0)	(1, 1)
$(P2L, P)$	(1, 0)	(0, 0)
$(P2L, L)$	(0, 0)	(1, 0)
$(L2C, L)$	(1, 0)	(0, 0)
$(L2C, C)$	(0, 0)	(1, 0)
(mea, C)	(1, 1)	(1, 2)
(exp, C)	(1, 2)	(1, 3)
$(S2UR, S)$	(1, 3)	(0, 0)
$(C2U, U)$	(0, 0)	(1, 0)
$(U2D, U)$	(1, 0)	(0, 0)
$(U2D, D)$	(0, 0)	(1, 0)
$(L2P, L)$	(1, 0)	(0, 0)
$(L2P, P)$	(0, 0)	(1, 0)
$(D2U, D)$	(1, 0)	(0, 0)
$(D2U, U)$	(0, 0)	(1, 0)
$(P2U, P)$	(1, 0)	(0, 0)
$(P2U, U)$	(0, 0)	(1, 0)
$(U2P, U)$	(1, 0)	(0, 0)
$(U2P, P)$	(0, 0)	(1, 0)
$(D2L, D)$	(1, 0)	(0, 0)
$(D2L, L)$	(0, 0)	(1, 0)
$(L2D, L)$	(1, 0)	(0, 0)
$(L2D, D)$	(0, 0)	(1, 0)
$(C2L, C)$	(1, 3)	(0, 0)
$(C2L, L)$	(0, 0)	(1, 0)

$mtype : \mathcal{T}_3 \rightarrow \{'process', 'transport'\}$ defined as follows:

$mtype(t) =$			
			$\lvert I_3(t)\rvert = 1$
		$\wedge$	$(\exists c : c \in I_3(t)$
$'process'$	if		$: Cb_3(t, c).0 = 1 \wedge Ce_3(t, c).0 = 1$
			$\wedge\, Cb_3(t, c).1 < Ce_3(t, c).1$
			$)$
			$\lvert I_3(t)\rvert = 2$
		$\wedge$	$(\exists c, c', n : c, c' \in I_3(t), n \in \mathbb{N}$
$'transport'$	if		$: Cb_3(t, c).0 = n \wedge Cb_3(t, c').0 = 0$
			$\wedge\, Ce_3(t, c).0 = 0 \wedge Ce_3(t, c').0 = n$
			$)$

Note that for this restricted case with only two types of meta-tasks the type identification could be a function returning a boolean value.

The meta-task definition elements outline the manufacturing possibilities of the system, together with the system dependent elements.

5.4 Instantiation of an exception recovery

Although the number of possible exceptional states of a complex manufacturing machine is very large, the number of meta-tasks is limited: 14 for the example. This section explains how this compact TRS definition level 3 can be used to instantiate a recovery, once an exception is encountered. The instantiation transformation is split in two steps: a search step and a conversion step. A search algorithm searches for traces of instantiated meta-tasks that bridge the gap between a given (exceptional) system state and a certain desired target system state. The conversion converts such a trace into an instantiated unselected TRS definition of the recovery. After instantiation of the recovery definition it can be merged with the remainder of the original definition. After scheduling of the resulting definition (see Chapter 3, [10]), manufacturing can be resumed.

5.4.1 Search algorithm

The search algorithm generates a search graph, starting from the exceptional system state and aiming to find a recovered target state. The nodes of the graph can be associated with system states, and the edges with instantiated meta-tasks or tasks. An instantiated meta-task can be characterized by the meta-task and the involved material instances. A trace in the search graph can be characterized by a sequence of instantiated meta-tasks: $(\mathcal{T}_3 \times \mathcal{P}(\mathcal{M}))^*$.

Let function Cm: $\mathcal{C} \rightarrow \mathbb{N}$ be a function that determines the number of material instances that can reside on all resources of a certain capability together.

$$Cm(c) = \sum_{r \in A(c)} Rm(r) \tag{5.2}$$

Let function E: $(\mathcal{C} \rightarrow \mathcal{P}(\mathcal{M} \times \mathcal{S}_m)) \rightarrow \mathcal{P}(\mathcal{T}_3 \times \mathcal{P}(\mathcal{M}))$ be a function that determines which tasks including the involved material instances are eligible to be instantiated in a certain system state:

$$\begin{array}{lll}
E(Ssys) = & & \\
\{\ (t, mr) \mid t \in \mathcal{T}_3, mr \subseteq \mathcal{M} & & \\
\ , mtype(t) =' process' \Longrightarrow & (\exists c, m, s_m & : c \in I_3(t), m \in mr, (m, s_m) \in Ssys(c) \\
 & & : s_m \geq Cb_3(t, c).1 \\
 &) & \\
\ , mtype(t) =' transport' \Longrightarrow & (\exists c, c' & : c, c' \in I_3(t) \\
 & & , Cb_3(t, c).0 = |mr|, Ce_3(t, c').0 = |mr| \\
 & & , mr \subseteq \{ms.0 | ms \in Ssys(c)\} \\
 & & : |Ssys(c')| \leq Cm(c') - |mr| \\
 &) & \\
\}
\end{array} \tag{5.3}$$

Process-type tasks are eligible to be instantiated if there is a material instance at the involved capability that is at least at the required manufacturing phase. Transport-type tasks are eligible to be instantiated if there is enough material available at the 'from' capability (c) and enough room at the 'to' capability (c').

Let function $updatestate$: $(\mathcal{C} \rightarrow \mathcal{P}(\mathcal{M} \times \mathcal{S}_m)) \times (\mathcal{T}_3 \times \mathcal{P}(\mathcal{M})) \rightarrow (\mathcal{C} \rightarrow \mathcal{P}(\mathcal{M} \times \mathcal{S}_m))$ be a function that updates the system state given an instantiated meta-task defined by

$updatestate(Ssys, (t, mr)) = Ssys'$ such that:

$$
\begin{array}{lll}
 & (mtype(t) =' process' \Longrightarrow & (\forall c, m, s_m \\
 & & : c \in I_3(t), m \in mr, (m, s_m) \in Ssys(c) \\
 & & : (m, s_m) \notin Ssys'(c) \wedge (m, Ce_3(t, c).1) \in Ssys'(c) \\
 & &) \\
 &) & \\
\wedge & (mtype(t) =' transport' \Longrightarrow & (\forall c, c', msr \\
 & & : c, c' \in I_3(t), Cb_3(t, c).0 = |mr|, Ce_3(t, c').0 = |mr| \\
 & & , msr = \{ms | ms \in Ssys(c), ms.0 \in mr\} \\
 & & : msr \not\subseteq Ssys'(c) \\
 & & \wedge (\forall ms : ms \in msr \\
 & & \quad : (ms.1 = Cb_3(t, c).1 \Rightarrow ms \in Ssys'(c')) \\
 & & \quad \wedge (ms.1 < Cb_3(t, c).1 \\
 & & \quad \Rightarrow (ms.0, Ce_3(t, c').1) \in Ssys'(c') \\
 & & \quad) \\
 & &) \\
 & &) \\
 &) &
\end{array}
\tag{5.4}
$$

In case of a process meta-task, the manufacturing phase of the involved material is updated. In case of a transport meta-task, the material involved is removed from the 'from' capability (c), and added to the 'to' capability (c'). The manufacturing phase of the material involved remains unchanged if the actual manufacturing phase was equal to the manufacturing phase at the beginning(pre-condition), and is reset to the end manufacturing phase (post-condition) if not.

The search algorithm should fulfill some requirements to fit in SMC. Without an ad-hoc search algorithm, an exception for which no recovery is predefined would imply an operator intervention. The machine's down time including manual recovery typically takes a few hours. Wafer processing cycle time is two orders of magnitude less, as can be concluded from Fig. 5.5. Automatic recovery by SMC is likely to take an amount of time which is of the same order. From this, it can be concluded that finding a recovery is most important, rather than finding a time-optimal (fastest) recovery. However, in general it is also possible that an exception cannot be recovered automatically as some essential meta-tasks are simply not under control of SMC.

A breadth-first search algorithm is applied, with a maximum search depth (*max_d*) to avoid wasting time by fruitless searching. By default, searching stops when the target state is reached for the first time. The time needed for recovery is expected to be reasonable, as the path consists of a minimum number of tasks. To analyse recovery time optimization possibilities, it is possible to extend the default search procedure to allow longer recovery traces than the one found first. The number of additional tasks is parameterized by variable (*max_d_diff*). In Fig. 5.6 the search algorithm is shown. In step 1, all eligible tasks are determined using function *E*. In step 2, the data sets describing the nodes (states) and edges (instantiated meta-tasks) are updated using function *updatestate*. For each node, the search depth at first visit is recorded. Some pruning rules are applied to reduce search graph growth:

- New edges are added only if the state reached is new or if the actual search depth minus the depth at first visit of the reached state does not exceed the maximum

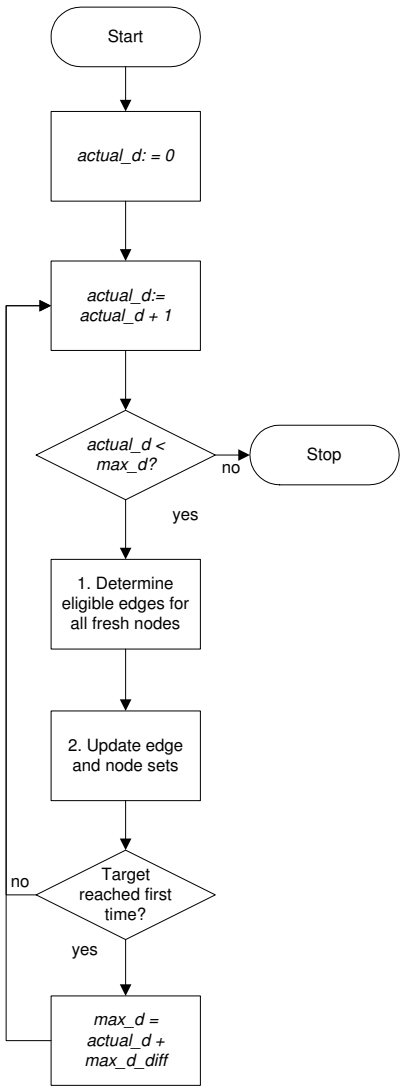


Figure 5.6: Search algorithm

depth difference (*max_d_diff*).

- Edges that result in non FIFO machine behavior are not added.
- Edges that close two-node cycles (from state b to a after an edge from state a to b) are not added.

  If necessary, more pruning rules can be added.

If the target state is reached the first time, the maximal search depth is revised such that searching stops after *max_d_diff* more iterations.

5.4.2 Conversion of a search trace into an instantiated, unselected TRS

With a successful recovery trace tr from the search graph, a definition of level 2 can be generated. Function *convert*: $(\mathcal{T}_3 \times \mathcal{P}(\mathcal{M}) \times \mathcal{T}_2)^* \times (\mathcal{T}_3 \times \mathcal{P}(\mathcal{M}))^* \rightarrow \mathcal{D}_2$ is a function that performs this conversion.

$$
\begin{array}{l}
convert(maptr, tr) = \\
\left\{ \begin{array}{ll}
D_2^{\varepsilon} & \text{if } tr = \varepsilon \\
convert(maptr +\!\!+ [hd(tr).0, hd(tr).1, t_2], tl(tr)) \cup D_2' & \text{if } tr \neq \varepsilon
\end{array} \right.
\end{array} \tag{5.5}
$$

Above, D_2^{ε} denotes the empty system definition, ε denotes the empty trace, and $(t_2, D_2') = addtask(maptr, hd(tr))$. To convert a found recovery trace tr into a recovery system definition D_2^{rec}, the *convert* function is called with (ε, tr) as argument. The internal variable $maptr$ is used to record the mapping of the instantiated meta-tasks to tasks and to determine the task precedence relation.

Function $addtask$: $(\mathcal{T}_3 \times \mathcal{P}(\mathcal{M}) \times \mathcal{T}_2)^* \times (\mathcal{T}_3 \times \mathcal{P}(\mathcal{M})) \rightarrow (\mathcal{T}_2 \times \mathcal{D}_2)$ is a function that instantiates for an instantiated metatask t_3 a task t_2 including its involved material mr to form an extension for the system definition of level 2, given a map trace $maptr$ defined by $addtask(maptr, (t_3, mr)) = (t_2, D_2')$ such that:

$$
\begin{array}{ll}
 & t_2 \in \mathcal{T}_2' \wedge t_2 \notin \mathcal{T}_2 \\
\wedge & (\forall c : c \in I_3(t_3) : c \in I_2'(t_2)) \\
\wedge & (\forall c : c \in I_3(t_3), Cb_3(t_3, c).0 \neq 0 : Cb_2'(t_2, c) = mr) \\
\wedge & (\forall c : c \in I_3(t_3), Ce_3(t_3, c).0 \neq 0 : Ce_2'(t_2, c) = mr) \\
\wedge & \begin{array}{l}
(\forall m, mtr \quad : m \in mr, mtr = filtermtr(maptr, m) \\
\qquad\qquad\quad : (hr(mtr).2, t_2) \in P_2' \\
) \\
(\forall c, ctr \quad : c \in I_3(t_3), ctr = filterctr(maptr, c) \\
\qquad\qquad\quad : capprec(ctr, c, t_2, mr) \subseteq P_2' \\
)
\end{array}
\end{array} \tag{5.6}
$$

Above, function $filterctr$:$(\mathcal{T}_3 \times \mathcal{P}(\mathcal{M}) \times \mathcal{T}_2)^* \times \mathcal{C} \rightarrow (\mathcal{T}_3 \times \mathcal{P}(\mathcal{M}) \times \mathcal{T}_2)^*$ filters the tasks involving some capability c from a trace $maptr$, and is defined as follows:

$$
\begin{array}{l}
filterctr(maptr, c) = \\
\left\{ \begin{array}{ll}
\varepsilon \text{if } maptr = \varepsilon & \\
filterctr(tl(maptr), c) & \text{if } maptr \neq \varepsilon \wedge c \notin I_3(hd(maptr).0) \\
[hd(maptr)] +\!\!+ filterctr(tl(maptr), c) & \text{if } maptr \neq \varepsilon \wedge c \in I_3(hd(maptr).0)
\end{array} \right.
\end{array} \tag{5.7}
$$

In addition, function $filtermtr$:$(\mathcal{T}_3 \times \mathcal{P}(\mathcal{M}) \times \mathcal{T}_2)^* \times \mathcal{M} \rightarrow (\mathcal{T}_3 \times \mathcal{P}(\mathcal{M}) \times \mathcal{T}_2)^*$ filters the tasks involving some material m from a trace $maptr$, and is defined as follows:

$$
\begin{array}{l}
filtermtr(maptr, m) = \\
\left\{ \begin{array}{ll}
\varepsilon & \text{if } maptr = \varepsilon \\
filtermtr(tl(maptr), m) & \text{if } maptr \neq \varepsilon \wedge m \notin hd(maptr).1 \\
[hd(maptr)] +\!\!+ filtermtr(tl(maptr), m) & \text{if } maptr \neq \varepsilon \wedge m \in hd(maptr).1
\end{array} \right.
\end{array} \tag{5.8}
$$

Function $addtask$ adds an additional task t_2 to the existing definition D_2, and assigns to this task the involved capabilities and the material instances to its begin and end material configuration. Furthermore, a precedence edge is instantiated from the rear task of the material lives of the involved material instances to the new task. Finally, precedence edges can be instantiated from tasks involving the same capabilities. This is done using function $capprec$: $(\mathcal{T}_3 \times \mathcal{P}(\mathcal{M}) \times \mathcal{T}_2)^* \times \mathcal{C} \times \mathcal{T}_2 \times \mathcal{P}(\mathcal{M}) \rightarrow \mathcal{P}(P_2)$, which is

defined as follows:

$$capprec(ctr, c, t_2, mr) = \begin{cases} \emptyset & \text{if } ctr = \varepsilon \\ \{(hr(ctr).2, t_2)\} & \text{if } ctr \neq \varepsilon \wedge |mr \cup hr(ctr).1| > Cm(c) \\ capprec(tr(ctr), c, t_2, mr \cup hr(ctr).1) & \text{if } ctr \neq \varepsilon \wedge |mr \cup hr(ctr).1| \leq Cm(c) \end{cases} \tag{5.9}$$

If the material capacity of an involved capability is equal to the involved number of materials, a precedence edge from the latest task in the trace for that capability involving another material is instantiated. For instance: a task involving the load robot capability will be linked to the latest task in the map trace that also involves the load robot and another material. If there is more room for material, as many material instances are skipped as there is room for. For instance: a task involving the wafer stage capability and material W3 will not be linked to the tasks involving another material W2, but will be linked to the latest task involving the wafer stage and yet another material W1. Note that some redundant precedence edges result from *capprec*, but they don't hurt.

If recovery brings the system in the same state as was expected without the exception, the remainder of the original scheduling problem can be resumed after recovery. Function *merge* merges the recovery TRS definition D_2^{rec} and the remainder of the initial TRS definition D_2^{ini} by linking the sub-lives of the involved material instances:

$merge(D_2^{rec}, D_2^{ini}) = D_2'$ such that $D_2' = D_2^{rec} \cup D_2^{ini}$ where $D_2^{rec} \cup D_2^{ini}$ is a pairwise union of all set definition elements [1] except that the material sub-lives are connected:

$$\begin{array}{lll} (\forall m, t, t' & : & m \in M_2^{rec}, t \in T_2^{rec}, t' \in T_2^{ini} \\ & , & (\nexists t'' : t'' \in T_2^{rec} : (t, t'') \in P_{2m}(D_2^{rec}, m)) \\ & , & (\nexists t'' : t'' \in T_2^{ini} : (t'', t') \in P_{2m}(D_2^{ini}, m)) \\ & : & (t, t') \in P_2' \\ ) & & \end{array} \tag{5.10}$$

Here, it is assumed that the nodes in the two system definitions do not intersect: $N_2^{rec} \cap N_2^{ini} = \varnothing$. This might imply renaming of nodes in the recovery. Furthermore, P_{2m} is a function describing for a material $m \in \mathcal{M}_2$ in a TRS definition $D_2^e \in \mathcal{D}_2$, a precedence relation between related nodes without redundant edges:

$$\begin{array}{ll} P_{2m}(D_2^e, m) = & \{(t, t') \\ & \mid (t, t') \in P_2^e \\ & \wedge \{cm.0 | cm \in Ce_2(t), m \in cm.1\} = \{cm.0 | cm \in Cb_2(t'), m \in cm.1\} \\ & \wedge \neg redundant(t, t', P_2^e) \\ & \} \end{array} \tag{5.11}$$

Where function *redundant*: $\mathcal{T}_2 \times \mathcal{T}_2 \times P_2 \rightarrow \mathbb{B}$ determines whether a precedence edge (t, t') is redundant in a precedence relation P:

$$redundant(t, t', P) = (\exists t'' : t'' \in \mathcal{T}_2, t'' \neq t, t'' \neq t' : path(t, t'', P) \wedge path(t'', t', P)) \tag{5.12}$$

Function *path*: $\mathcal{N}_2 \times \mathcal{N}_2 \times P_2 \rightarrow \mathbb{B}$ used above determines whether there is a path between two tasks t and t' in a precedence relation P:

$$path(t, t', P) = \begin{cases} true & \text{if } t = t' \\ (\exists t'' : (t, t'') \in P : path(t'', t', P)) & \text{if } t \neq t' \end{cases} \tag{5.13}$$

[1] As functions can be seen as sets, $\cup$ on functions is defined analogously to $\cup$ on sets.

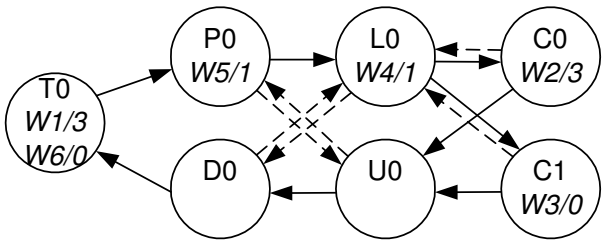


Figure 5.7: Situation in the machine after the exception

5.5 Results

To illustrate the approach, an example exception in the wafer scanner is considered: the measure task of wafer W3 fails, which is raised at the end of the task at t = 91 [sec] . After completion of the tasks in progress (pre-alignment of W5), at $t = 94$ [sec], the situation in the machine is as depicted in Fig. 5.7.

The exception has caused wafer W3 to stick to manufacturing phase 1 ('pre-aligned') rather than manufacturing phase 2 ('measured'). A trivial recovery would be to retry the measure task of the wafer. However, for the purpose of this wafer it is assumed that retrying measuring is not useful without retrying pre-aligning. Therefore, the manufacturing phase of wafer W3 is set to 0 ('fresh'). The exceptional system state can be constructed from Fig. 5.7 in a straightforward way. The target system state differs from this state only for wafer W3: it should be in phase 2 ('measured').

First, the search algorithm is used to find a recovery without allowing longer traces: $max\_d\_diff = 0$. This results in a recovery trace consisting of 14 tasks. After conversion and merging the recovery with the remainder of the tasks to be done, the task precedence graph of Fig. 5.8 results. In this precedence graph, the recovery tasks per wafer are depicted horizontally, whereas the remainder of the original graph is unchanged: the remaining tasks per wafer are depicted vertically. This recovery can be characterized as a 'rewind' recovery: wafers W4 and W5 are parked at the Discharge unit and the Unload robot, respectively, to clear the path for wafer W3 to be transported back to the Pre-alignment unit. Assuming that recovery search and scheduling (SMC reaction) take 10 seconds, the schedule in Fig. 5.9 can be obtained.

This schedule finishes at $t = 296$ [sec], which is 81 seconds later than the original schedule. From this, the recovery itself took 68 seconds, as 13 seconds were needed for waiting till the machine was idle (3 [sec]) and SMC reaction (10 [sec]). It appears that recovery indeed takes an amount of time that is in the same order as wafer processing cycle time.

Subsequently, optimization possibilities are analyzed. The search algorithm is restarted with $max\_d = 14$ and $max\_d\_diff = 2$, to allow for traces up to 16 tasks. After cutting off unsuccessful traces, the search graph as depicted in Fig. 5.10 results. This search graph also contains a trace consisting of 16 tasks that can be characterized as a 'cycle' recovery. The precedence graph for this recovery and the remainder of the original graph is depicted in Fig. 5.11. Wafer W3, as well as wafers W4 and W5 follow an additional cycle through the machine to reach the pre-alignment unit again via the unload robot. A schedule of the cycle recovery is depicted in Fig. 5.12, assuming that SMC reaction still takes 10 seconds. Although this recovery uses more tasks, it is faster as more parallelism can be exploited. The end time of this schedule is 285 [sec], which means that the delay caused by recovery is 14 % less than the 'rewind' recovery. This is an over-approximation

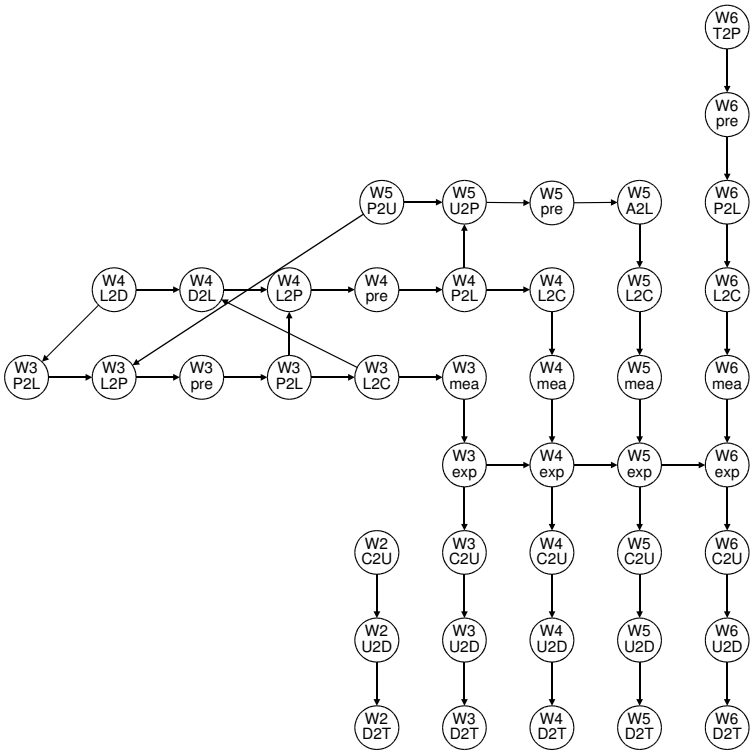


Figure 5.8: Precedence graph for 'rewind' recovery and remainder

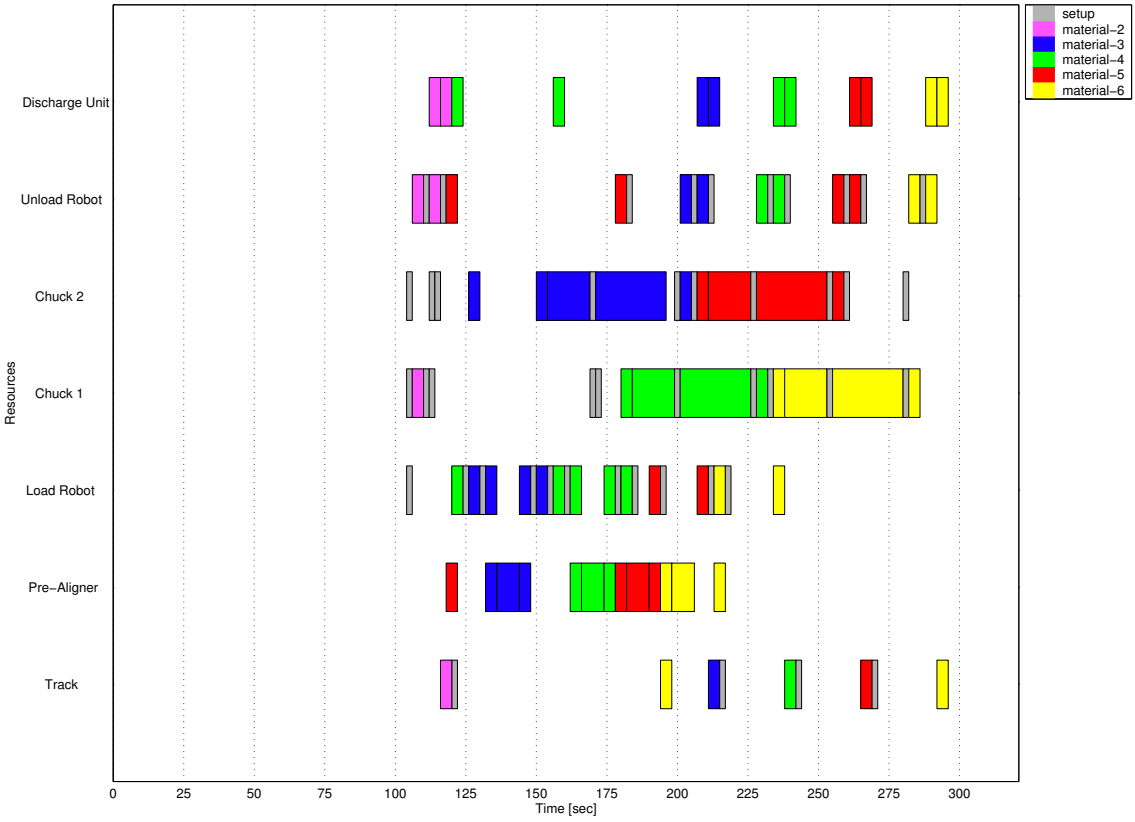


Figure 5.9: Schedule of rewind recovery

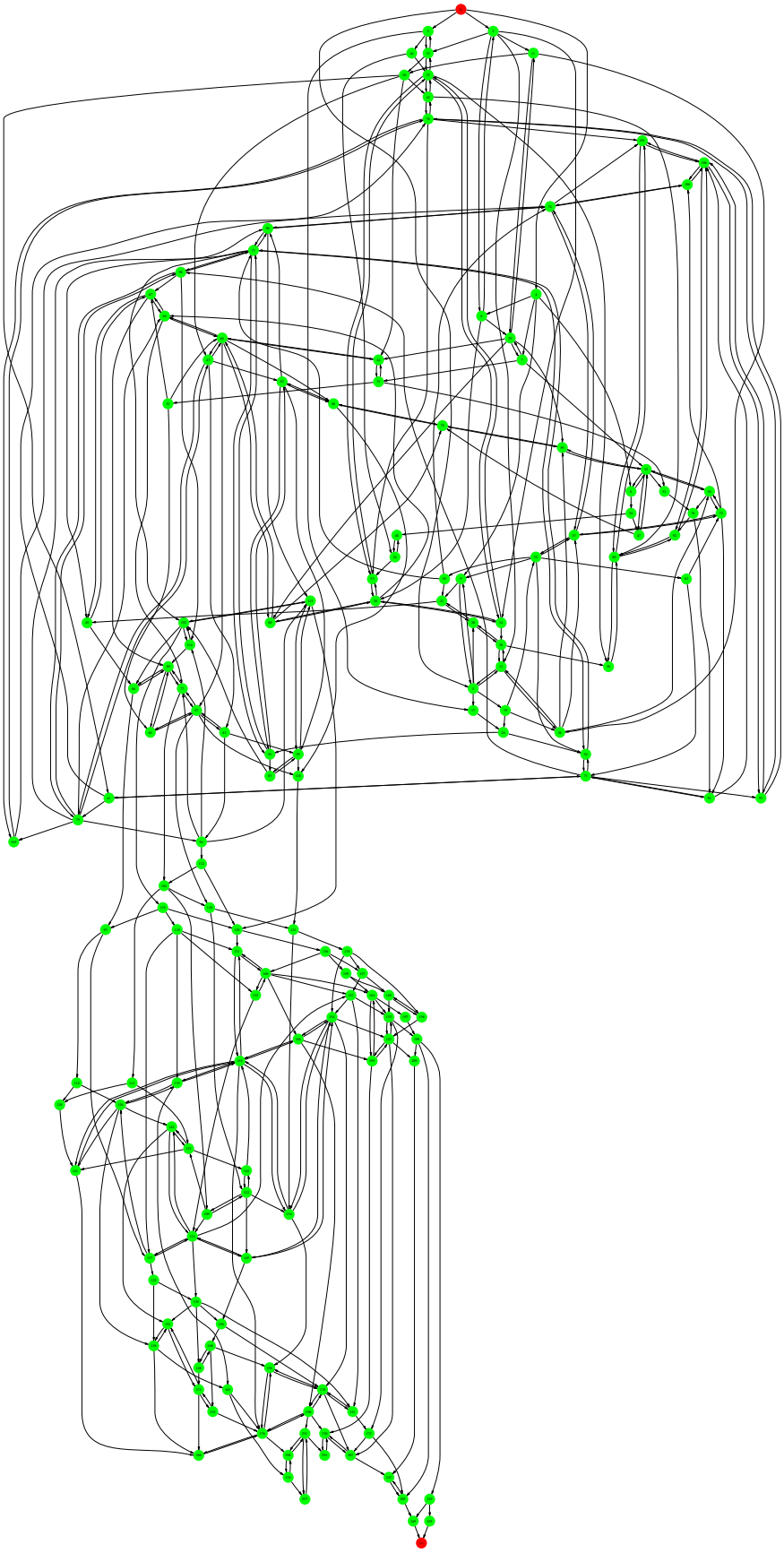


Figure 5.10: Search graph

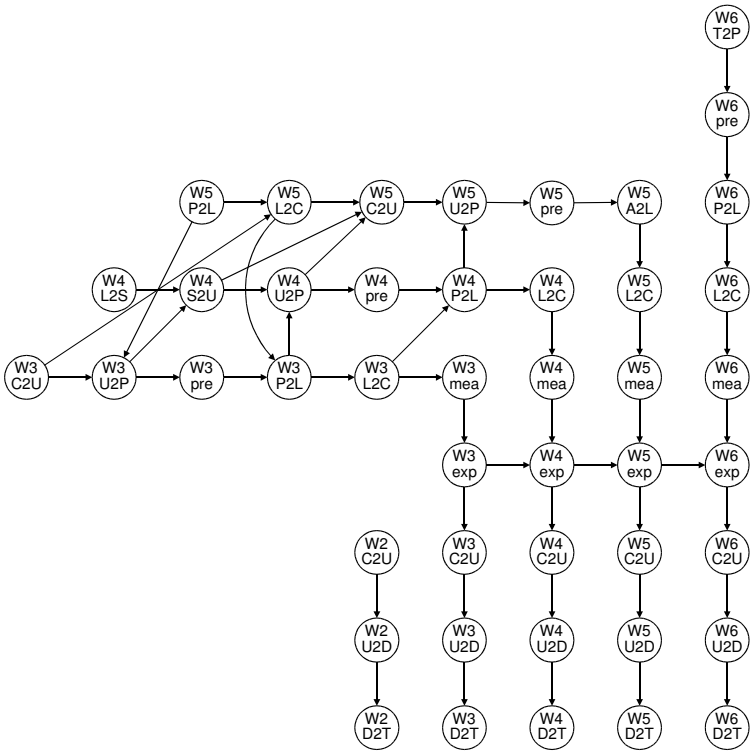


Figure 5.11: Precedence graph for 'cycle' recovery and remainder

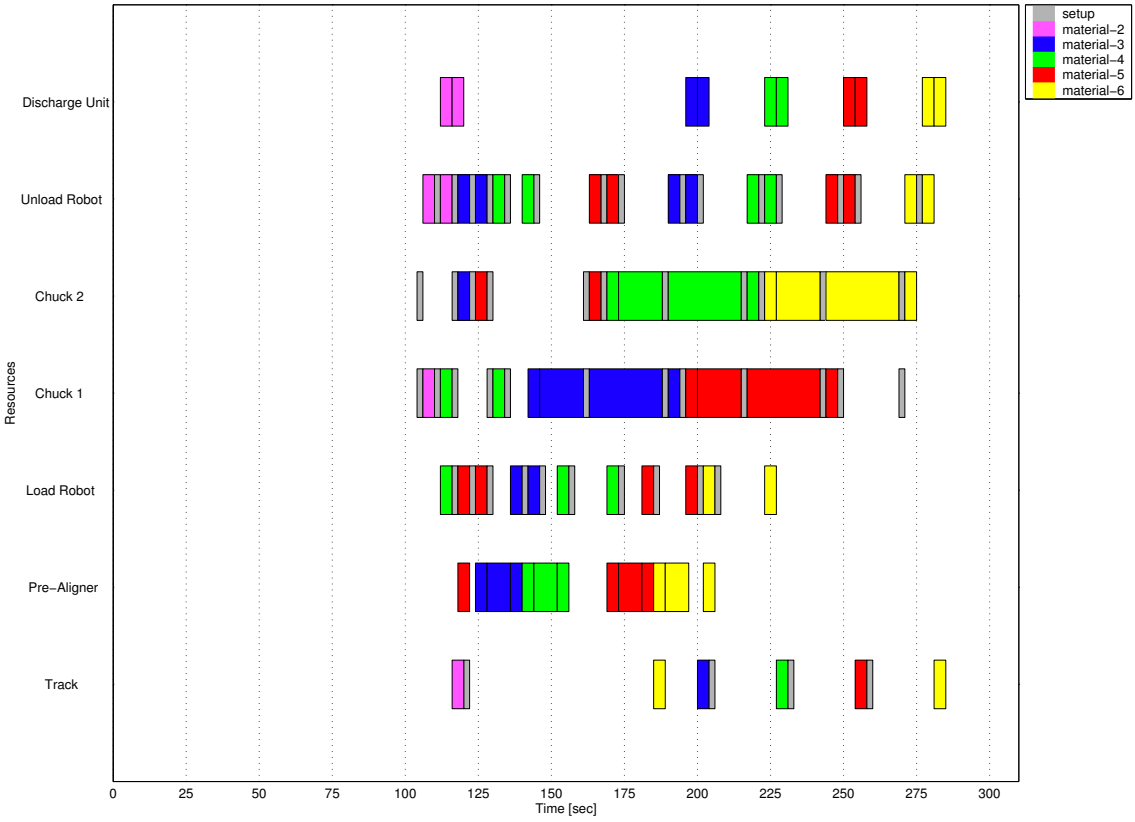


Figure 5.12: Schedule 1 of cycle recovery

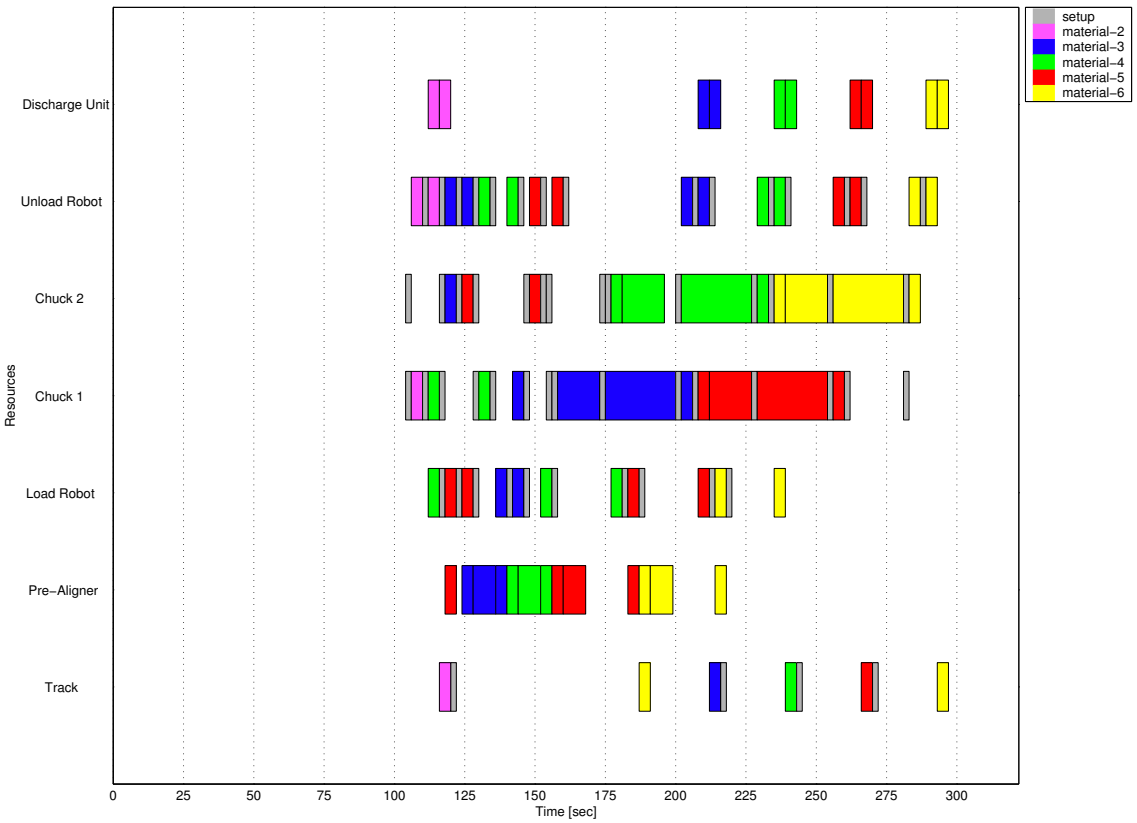


Figure 5.13: Schedule 2 of cycle recovery

as in reality SMC reaction is slowed down by the more extensive search, which could even cost more than optimization brings.

To show that multiple schedules are possible for one recovery, Fig. 5.13 is added, that shows another schedule for the 'cycle' recovery. This schedule ends at $t = 297$ [sec], which is even later than the schedule for the 'rewind' recovery found first. In this case, wafer W5 is transported from its chuck to the unload robot before wafer W3 is transported from the load robot onto a chuck, which was done the other way around in Fig. 5.12.

5.6 Conclusions

The huge number of exceptional states that a complex manufacturing machine can get in makes it practically impossible to predefine recoveries for each exception. This chapter describes an approach to overcome this problem by ad-hoc run time recovery search once a specific exception occurs.

The manufacturing possibilities can be defined intuitively within the TRS framework, in the form of an uninstantiated TRS definition. A search algorithm bridges the gap between the exceptional state and a target state, thus instantiating a recovery.

Results from an industrial case show that the algorithm is suited for practical use as the first recovery is found in reasonable time, and recovery speed optimization is not interesting. The recoveries found by the algorithm could be cached in a database by SMC to let it learn how to recover from exceptions.

The following open issues remain. Instead of the pragmatic maximum search depth to avoid fruitless searching, a more elegant approach would be to do a reachability analysis.

Furthermore, the application domain can be widened to cover not only transport and process meta-tasks, but e.g. also initialization or calibration meta-tasks.

References

[1] ASML, 2004. Information on wafer scanners available through URL `http://www.asml.com/`, item: products - lithography.

[2] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–341, 1994.

[3] H. Cho and R. A. Wysk. Graph-theoretic deadlock detection and resolution for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 11(3):413–421, 1995.

[4] M. P. Fanti and M. Zhou. Deadlock control methods in automated manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 34(1):5–22, 2004.

[5] I. C. M. Flinsenberg. *Route Planning Algorithms for Car Navigation.* PhD thesis, Eindhoven University of Technology, The Netherlands, September 2004.

[6] A. Garnaev. *Search games and other applications of game theory.* Springer, 2000.

[7] P. Gohari and W. M. Wonham. Reduced supervisors for timed discrete-event systems. *IEEE Transactions on Automatic Control*, 48(7):1187–1198, 2003.

[8] Y. Li and Z. H. Lin. Supervisory control of probabilistic discrete-event systems with recovery. *IEEE Transactions on Automatic Control*, 44(10):1971–1975, 1999.

[9] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, N. C. W. M. Braspenning, and J. E. Rooda. Predictive scheduling in complex manufacturing machines: scheduling alternatives and algorithm. *submitted to IEEE TAC.*

[10] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, N. C. W. M. Braspenning, and J. E. Rooda. Predictive scheduling in complex manufacturing machines: machine-specific constraints. *submitted to IEEE TSM.*

[11] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, and J. E. Rooda. Design of supervisory machine control. In K. Glover and J. Maciejowski, editors, *Proceedings of the European Control Conference 2003*, 2003. CD-ROM.

[12] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, 1995.

[13] R. G. Qiu and S. B. Joshi. A structured adaptive supervisory control methodology for modeling the control of a discrete event manufacturing system. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 29(6):573–586, 1999.

[14] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[15] H. J. Yoon and D. Y. Lee. Deadlock-free scheduling of photolithography equipment in semiconductor fabrication. *IEEE Transactions on Semiconductor Manufacturing*, 17(1):42–54, February 2004.

CHAPTER

SIX

KINEMATIC CALIBRATION SEQUENCING IN HIGH-PRECISION MACHINES

This chapter contains the paper *Kinematic Calibration Sequencing in High-Precision Machines* that has been protected in patent application ASML ref. P-1704. The idea has been filed with the first filing in Europe in October 2003, number 03256456.9. The paper text is in the subsequent filing of September and October 2004, which is also filed in Japan (number 2004-286595) and the US.

Kinematic calibration sequencing in high-precision machines

N.J.M. van den Nieuwelaar †*, J.M. van de Mortel-Fronczak †,
M.A.R. Stoets †, J.E. Rooda †

Abstract

In high-precision manufacturing machines, geometric imperfections in both machine hardware and product material need to be corrected for during production. To that end, sequences of calibration steps are performed consisting of measurements using different types of sensors and computations. In this chapter, a framework is presented that identifies the constraints for kinematic calibration sequences in high-precision machines. The framework can be applied to design calibration sequences. Moreover, it can be complemented with a search algorithm for run-time application in Supervisory Machine Control (SMC). The different calibration steps are specified by, so-called, meta-tasks having pre-conditions and post-conditions. Pre-conditions define in which machine state, in terms of geometric parameter inaccuracies, the meta-task can be executed successfully. Post-conditions define the machine state expected after execution of the meta-task, which in case of computations depends on the machine state before execution. This dependency is specified in terms of linear equations that can be deduced from the kinematic chains of the different sensor measurements. Linear transformations are used to predict the state transition resulting from a calibration step. Calibration sequences are transformed into scheduling problems to enable prediction of time performance. The framework is illustrated using an example calibration sequence from a wafer scanner.

6.1 Introduction

Machines manufacturing at nanometer accuracy (high-precision machines) have to correct for all kinds of geometric imperfections. Literature describes many kinematic chain-based approaches to correct for imperfections in the machine hardware [2, 3, 5, 6, 13, 14, 16, 17]. However, in high-precision machines, some complicating requirements play a role. One of them is that also imperfections originating from the material that is processed must be corrected for, as well as drift effects in the machine hardware. This implies that kinematic calibration is not only an off-line activity, but it must also be performed during production. Any on-line production activities are a responsibility of Supervisory Machine Control (SMC), which implies that kinematic calibration must be done autonomously: without help of an operator. Besides that, the kinematic calibration is under timing pressure: the longer the calibration takes the less productivity is obtained. Furthermore, the imperfections being faced are multiple orders of magnitude bigger than allowed. On the other hand, high-end sensors available in a high-precision machine have a limited capture range and typically have an inaccuracy that is only one order of magnitude smaller

† Eindhoven University of Technology: P.O. box 513, 5600 MB Eindhoven, The Netherlands.
* ASML: De Run 6501, 5504 DR Veldhoven, The Netherlands.
Corresponding author: N.J.M. van den Nieuwelaar, e-mail: n.j.m.v.d.nieuwelaar@tue.nl

than their capture range. This implies that calibration must consist of a sequence of steps combining multiple types of sensors, each having their own type of kinematic chain. The multiple sensors might make it possible to perform calibration steps in parallel.

A desirable property for SMC is predictability, as explained in Chapters 2 and 3 ([9, 10]). In the context of kinematic calibration, predictability implies predictable geometric parameter inaccuracy. This chapter describes a framework for analysis and generation of kinematic calibration sequences in, for instance, high-precision machines that can be embedded in SMC.

6.1.1 Layered task resource system framework

From an SMC point of view, a machine can be considered as a task resource system (TRS). Tasks can be associated with manufacturing processes, whereas resources can be associated with mechatronic systems. Transforming a manufacturing request into machine behavior can be structured in three phases. First, a scheduling problem must be instantiated from the manufacturing request, taking into account the limitations of the machine. This transformation is called *instantiating*. In the context of kinematic calibration this comes down to generation of a calibration sequence. Although the word 'sequence' might suggest otherwise, it may encompass task parallelism and therefore 'task graph' would be a more precise description. The structure of the resulting scheduling problem containing the task graph shows many similarities with the job shop scheduling problem [12]. The manufacturing process of a material instance can be associated with a job, whereas the different parallel mechatronic systems can be associated with the different machines in a job shop. Subsequently, resources must be assigned to the tasks in the instantiated scheduling problem in some order, taking into account the fact that resources are able to perform certain tasks only, and only one at a time. This transformation is called *selecting*. The selected order of tasks to be performed by selected resources may imply consecutive state transitions of those resources, which is analogous to the setup times for mode switching in job shop scheduling. Finally, start and finish times can be assigned to the tasks, taking into account the speed of the resources. This transformation is called *timing*. Combination of the selecting and timing transformation is referred to as scheduling, and scheduling in complex manufacturing machines is addressed in Chapters 2 and 3 ([9, 10]).

During the three transformation phases of instantiating, selecting and timing, choices must be made. The consequences of a choice in a certain transformation on the machine behavior can only be evaluated by performing the consecutive transformations. Therefore, a transformation phase strongly relies on information from subsequent phases. The layered TRS framework shown in Fig. 6.1 displays the hierarchically related transformation phases as functionality layers (A through C) and the different TRS definition levels (0 through 3) as interfaces between the layers (see Chapter 1 and [11]).

6.1.2 Structure of the chapter

The structure of this chapter is as follows. Throughout the chapter, we use a case of a calibration sequence in a high-precision machine for illustration: reticle stage align in a wafer scanner [1]. Section 6.2 introduces the issues involved in instantiating a calibration sequence using the wafer scanner case (layer C in Fig. 6.1) and explains how a calibration sequence can be defined as a scheduling problem (TRS level 2 in Fig. 6.1).

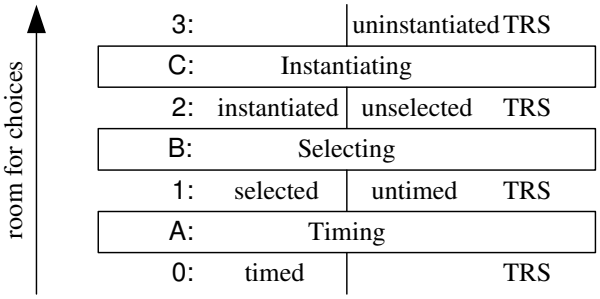


Figure 6.1: Layered Task Resource System (TRS) framework

Section 6.3 describes the relation between calibration parameter inaccuracies as a system of linear geometric relations. In Section 6.4, we define the constraints for instantiating a calibration sequence: the uninstantiated system definition (level 3) in Fig. 6.1. First, we consider parameter inaccuracies without variance. We define the system state, as well as the instantiation constraint parameters in the form of the TRS definition elements. Based on these constraint parameters, we define the instantiation constraints themselves. Subsequently, we describe the consequences of adding variance to the parameter inaccuracies. Finally, we discuss conversion of a calibration sequence into a scheduling problem. In Section 6.5, we illustrate the framework using the reticle stage align case from the wafer scanner. After definition of the example as an uninstantiated TRS (TRS level 3 in Fig. 6.1), we show the evolution of the parameter errors for an example sequence, based on which we verify the sequence. Furthermore, we analyse the timing performance of the sequence. Finally, we present some concluding remarks in Section 6.6.

6.2 Calibrating a wafer scanner

6.2.1 Reticle Stage Alignment

The primary manufacturing process of a wafer scanner is the exposure of a mask containing an IC pattern onto wafers. The mask is engraved on a so-called reticle that is situated at a reticle stage. Light projects the pattern on the reticle onto the wafer that is situated at a wafer stage. During exposure, the reticle stage and the wafer stage make a scanning movement. Before wafers can be exposed they must be measured to make correction for imperfections possible. The machine under consideration is a dual stage wafer scanner that is equipped with two wafer stages to enable parallel measurement and exposure of wafers. As a consequence, wafer stages need to be swapped from measure to expose position and vice versa. At each position, a separate relative position measuring system is used. After a swap, this measuring system must be zeroed, which is done using sensors with an inaccuracy in the order of micrometers. The resulting so-called *zeroing* error is too big to start exposure, which requires an inaccuracy in the order of nanometers. Therefore, an on-line calibration sequence called Reticle Stage Alignment (RSA) must be executed: the reticle stage carrying the reticle must be aligned versus the wafer stage carrying the wafer. In this chapter, we consider the levelling part of RSA that focusses on the deviation in wafer stage height and tilt.

In this calibration sequence two types of measurements can be applied, see Fig. 6.2.

A Confidence Sensor measurement (CS) determines the height of a point on the surface

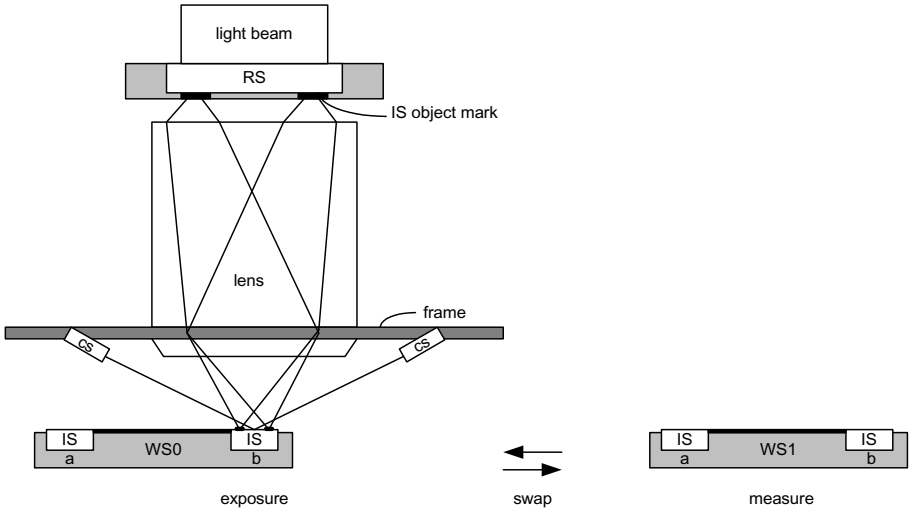
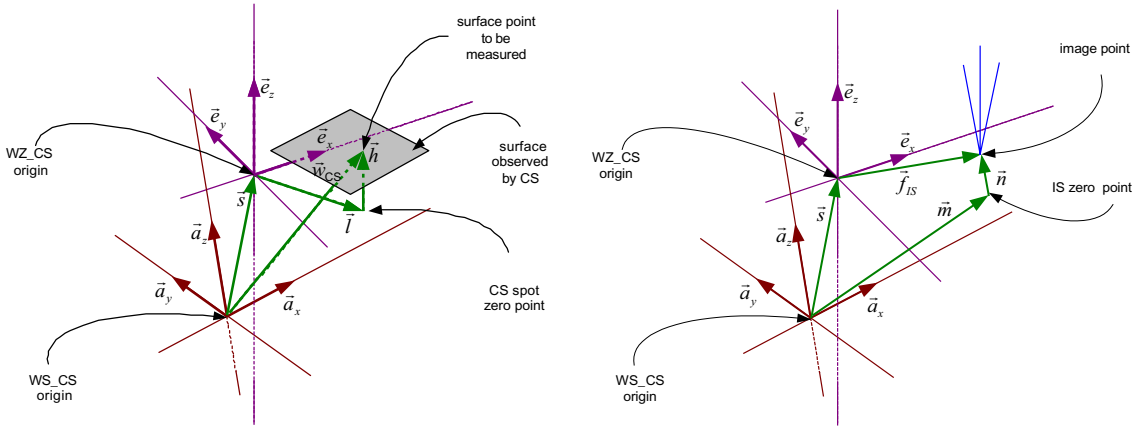


Figure 6.2: Overview of the main subsystems involved in Reticle Stage Alignment (RSA)

of a wafer stage (WS0 or WS1). In an Image Sensor (IS) measurement, IS object marks on the reticle stage (RS) are projected through the lens. Image sensors mounted on the wafer stage measure the aerial position of the projected object marks. Sensors can operate in different modes that are characterized by their specific capture range and inaccuracy. However, the different types of measurements involve different parameters. Each type of measurement is characterized by its own kinematic chain, as is depicted in Fig. 6.3.



(a) Confidence sensor

(b) Image sensor

Figure 6.3: Kinematic chains for (a) the confidence sensor (CS) and (b) the image sensor (IS)

Fig. 6.3(a) shows the kinematic chain for a CS measurement involving

- Wafer stage coordinate system (WS_CS) with base $\alpha = (\vec{a}_x, \vec{a}_y, \vec{a}_z)$.
- Wafer (metro)frame coordinate system or wafer stage zeroing coordinate system (WZ_CS) with base $\varepsilon = (\vec{e}_x, \vec{e}_y, \vec{e}_z)$.

- CS spot zero point.
- Surface point to be measured.

We can identify the following vectors in Fig. 6.3(a):

- $\vec{h}$: vector from the CS spot zero point to the surface point to be measured.
- $\vec{l}$: vector from the WZ_CS origin to the CS spot zero point.
- $\vec{s}$: vector from the WS_CS origin to the WZ_CS origin.
- $\vec{w}_{\mathrm{CS}}$: vector from the WS_CS origin to the surface point to be measured.

We derive the following vector relation representing te kinematic chain for a CS measurement:

$$\vec{s} + \vec{l} + \vec{h} = \vec{w}_{\mathrm{CS}}. \tag{6.1}$$

Fig. 6.3(b) shows the kinematic chain for an image sensor measurement involving

- WS_CS and WZ_CS as in the CS kinematic chain.
- Image sensor (IS) zero point.
- Image point.

We can identify the following vectors in Fig. 6.3(b):

- $\vec{s}$: as in the CS kinematic chain.
- $\vec{n}$: vector from the IS zero point to the image point.
- $\vec{f}_{\mathrm{IS}}$: vector from the WZ_CS origin to the image point.
- $\vec{m}$: vector from the WS_CS origin to the IS zero point.

When the wafer stage is in the aligned position, the IS zero point and the image point are at the same position. In that case, $\vec{n}$ reduces to zero and $\vec{s}$ is the aligned position vector. From Fig. 6.3(b) we derive the following vector relation representing the kinematic chain for an IS measurement:

$$\vec{f}_{\mathrm{IS}} + \vec{s} = \vec{m}. \tag{6.2}$$

To bridge the gap between the initial inaccuracy in the order of micrometers and desired accuracy in the order of nanometers, the results of both types of measurements must be combined. In between the different types of measurements, the geometric relation between the different parameters must be used for conversion.

6.2.2 A calibration sequence as a scheduling problem

In Chapter 3 ([10]), the scheduling model of a complex machine with its physical restrictions is defined. For the purpose of this chapter, the elements needed for the timing transformation (see Fig. 6.1), as well as material logistics are not relevant. Without them, the scheduling model can be defined by a 6–tuple $(\mathcal{T}_2, P_2, \mathcal{R}, \mathcal{C}, A, I_2)$:

- $\mathcal{T}_2$ is a finite set of elements called tasks, e.g., an exposure.
- $P_2 \subseteq \mathcal{T}_2 \times \mathcal{T}_2$ is the precedence relation between tasks, e.g., a RSA task precedes an exposure task.
- $\mathcal{C}$ is a finite set of elements called capabilities, e.g., wafer stage (WS).
- $\mathcal{R}$ is a finite set of elements called resources, e.g., wafer stage 0 and 1 (WS0, WS1).
- A: $\mathcal{C} \rightarrow \mathcal{P}(\mathcal{R})$ gives the set of resources that are available for a certain capability, e.g., resources WS0 and WS1 for capability WS.
- I_2: $\mathcal{T}_2 \rightarrow \mathcal{P}(\mathcal{C})$ gives the set of capabilities that are involved in a certain task, e.g., an exposure task involves WS and RS.

6.3 System of linear geometric relations

In this section, we compose and explain a system of linear geometric relations between parameter inaccuracies using simple examples concerning the determination of the position of a plane. Subsequently, we apply the resulting relations to the reticle stage alignment case.

6.3.1 Parameter relations

As discussed in the previous section, a high-precision machine is equipped with different types of sensors that are able to perform different types of measurements. The parameters involving some type of measurement can be related using a linear equation. The equations for different types of measurements can be defined using matrices:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{6.3}$$

Here, $\mathbf{b}$ contains known constants, $\mathbf{x}$ contains parameters and $\mathbf{A}$ contains coefficients. Each type of measurement can be done at a limited number of measurement positions and sensor units. For different measurement positions and sensor units, different equations are applicable. If we instantiate (6.3) for these, we get

$$\mathbf{A}_\mathrm{p}\mathbf{x}_\mathrm{p} = \mathbf{b}_\mathrm{p} \tag{6.4}$$

Example 1 (Plane positioning) *We consider a plane nearly parallel to the XY-plane of a coordinate system with base $\alpha = (\vec{a}_x, \vec{a}_y, \vec{a}_z)$ and origin O_α (see Fig. 6.4). A sensor is available to measure the height (z) of points on the plane. We derive the following linear relation of the parameters involved in the height measurement:*

$$s^\alpha_{z_p} = s^\alpha_{z_0} + R_y s^\alpha_{x_p} + R_x s^\alpha_{y_p}. \tag{6.5}$$

We ran rewrite this in the form of (6.3) *as follows:*

$$s^{\alpha}_{z_0} + R_y s^{\alpha}_{x_p} + R_x s^{\alpha}_{y_p} - s^{\alpha}_{z_p} = 0. \tag{6.6}$$

Suppose that the sensor can measure at three different positions in the XY-plane: ($s^{\alpha}_{x_p}$, $s^{\alpha}_{y_p}$) = *(1, 0), (0, 1), and (1,1), (see Fig. 6.4). Instantiating the three positions in* (6.6) *yields:*

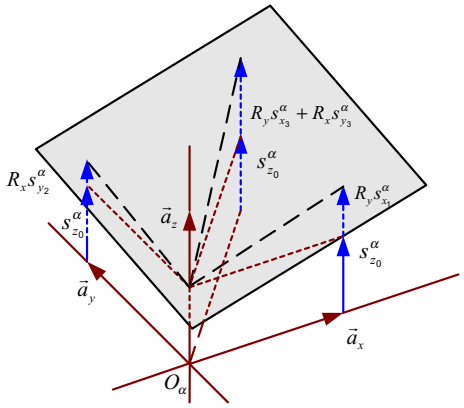


Figure 6.4: Plane positioning using three height measurements

$$\begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 \\ 1 & 1 & 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} s^{\alpha}_{z_0} \\ R_y \\ R_x \\ s^{\alpha}_{z_1} \\ s^{\alpha}_{z_2} \\ s^{\alpha}_{z_3} \end{bmatrix} = \mathbf{0}. \tag{6.7}$$

Using m equations from a linear system such as (6.4), it is possible to derive r unknown parameters. Here, r denotes the rank of the coefficient matrix for the m equations, i.e., the number of linearly independent parameters. If more than r parameters are unknown, a solution can be to combine unknown parameters into combined parameters. A combined parameter can be defined as some linear combination of original parameters. Substitution of such combined parameters in the m equations may result in a solvable system. The solution encompasses some original parameters and the combined parameters, thus localizing the solution freedom in the combined parameters. Using this result it might be possible to execute a subsequent calibration task if the combined parameters are not needed as an input, or if the combined parameters suffice as input. The equations defining the combined parameters in terms of the original parameters can be of use to determine the original parameters later on. Therefore, we add the combined parameters and the equations defining them to (6.4), together with the equations resulting from substitutions, yielding:

$$\mathbf{A}_{\text{pc}}\mathbf{x}_{\text{pc}} = \mathbf{b}_{\text{pc}}. \tag{6.8}$$

Note that numerous combined parameters and substitutions can be thought of. Furthermore, numerous subsets of equations from (6.8) can be thought of to derive numerous subsets of parameters from. We limit ourselves to providing constraints for combining and substituting parameters.

Example 2 (Combining parameters) *Suppose we can measure the plane height at two positions* $s_4^\alpha(x,y) = (-1,-1)$ *and* $s_5^\alpha(x,y) = (1,1)$ *yielding* $s_{z_4}^\alpha$ *and* $s_{z_5}^\alpha$*. Then, the system describing the parameter relations yields:*

$$\begin{bmatrix} 1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} s_{z_0}^\alpha \\ R_y \\ R_x \end{bmatrix} = \begin{bmatrix} s_{z_4}^\alpha \\ s_{z_5}^\alpha \end{bmatrix}. \tag{6.9}$$

The rank of the coefficient matrix indicates that we can only solve (6.9) *for two parameters. Therefore, we define a combined parameter*

$$R_{xy} = R_x + R_y, \tag{6.10}$$

and substitute this combined parameter into (6.9) *to obtain the following system with full rank:*

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} s_{z_0}^\alpha \\ R_{xy} \end{bmatrix} = \begin{bmatrix} s_{z_4}^\alpha \\ s_{z_5}^\alpha \end{bmatrix}. \tag{6.11}$$

Now, the tilt of the plane can be determined versus the line $x = y$*. Adding Equations* (6.10, 6.11) *to* (6.9) *results in the following system of geometric relations:*

$$\begin{bmatrix} 1 & -1 & -1 & 0 & -1 & 0 \\ 1 & 1 & 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 & -1 & 0 \\ 1 & 0 & 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} s_{z_0}^\alpha \\ R_x \\ R_y \\ R_{xy} \\ s_{z_4}^\alpha \\ s_{z_5}^\alpha \end{bmatrix} = \mathbf{0}. \tag{6.12}$$

6.3.2 Parameter error relations

We define an estimate $\tilde{x}$ of parameter x with error Δx as follows:

$$\tilde{x} \equiv x + \Delta x. \tag{6.13}$$

In certain circumstances, it is allowed to assume that parameter coefficients can be estimated accurately enough to neglect their errors, i.e.,

$$\tilde{\alpha} \equiv \alpha + \Delta\alpha \approx \alpha. \tag{6.14}$$

In the plane example this is the case as the measured plane is nearly parallel to the XY-plane and the measurement positions lie relatively far apart. Therefore, the height and tilt parameters are relatively insensitive for small deviations in the measurement position coefficients. If the role of parameters and coefficients would be exchanged, the assumption would not be justified. In that case, the height and tilt 'coefficients' would form the basis to determine the (x,y) position 'parameters' of the sensors. In that case, the (x,y) position is relatively sensitive for small deviations in the tilt of the plane.

If the circumstances mentioned above apply, linear system (6.8) can be written in terms of parameter errors as follows:

$$\begin{aligned} \mathbf{A}_{\text{pc}}\tilde{\mathbf{x}}_{\text{pc}} &= \tilde{\mathbf{b}}_{\text{pc}}, \\ \mathbf{A}_{\text{pc}}(\mathbf{x}_{\text{pc}} + \Delta\mathbf{x}_{\text{pc}}) &= \mathbf{b}_{\text{pc}} + \Delta\mathbf{b}_{\text{pc}}. \end{aligned} \tag{6.15}$$

Subtracting exact relation (6.8) from estimate relation (6.15) yields the relation between parameter errors:

$$\mathbf{A}_{\mathrm{pc}}\Delta\mathbf{x}_{\mathrm{pc}} = \Delta\mathbf{b}_{\mathrm{pc}}. \tag{6.16}$$

We can conclude that in certain circumstances, both parameter values and parameter errors are equally related. For convenience, we omit Δ in the remainder of this chapter.

The parameter errors show some distribution. We refer to such a parameter error distribution as parameter inaccuracy in the remainder of this chapter. To determine whether parameters are in capture range of a sensor, or whether a calibration sequence results in the required inaccuracy, the limit of the absolute error is of importance. A straightforward approach to express inaccuracy is to consider the worst case error bound. A more realistic approach is a statistical approach, using a distribution that is characterized by a mean and a variance. In whatever form the inaccuracy is expressed, the linear system (6.8) composed in the previous subsection can be used for deriving parameter inaccuracies from other parameters inaccuracies: computation tasks. Using measurements, parameters (or their inaccuracy) can be obtained directly. We can extend (6.8) for deriving parameter inaccuracies resulting from measurements by adding sensor inaccuracies to $\mathbf{b}_{\mathrm{pc}}$ and rows indicating the measured parameters to $\mathbf{A}_{\mathrm{pc}}$

$$\mathbf{A}_{\mathrm{pcm}}\mathbf{x}_{\mathrm{pc}} = \mathbf{b}_{\mathrm{pcm}}. \tag{6.17}$$

Example 3 (Sensor inaccuracies) *In the plane positioning example, we measure the height $s^{\alpha}_{z_p}$ of a point p in the plane with a certain inaccuracy β_p. Adding the sensor inaccuracies for the two measurements yields the following system:*

$$\begin{bmatrix} 1 & -1 & -1 & 0 & -1 & 0 \\ 1 & 1 & 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 & -1 & 0 \\ 1 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s^{\alpha}_{z_0} \\ R_y \\ R_x \\ R_{xy} \\ s^{\alpha}_{z_4} \\ s^{\alpha}_{z_5} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \beta_4 \\ \beta_5 \end{bmatrix}. \tag{6.18}$$

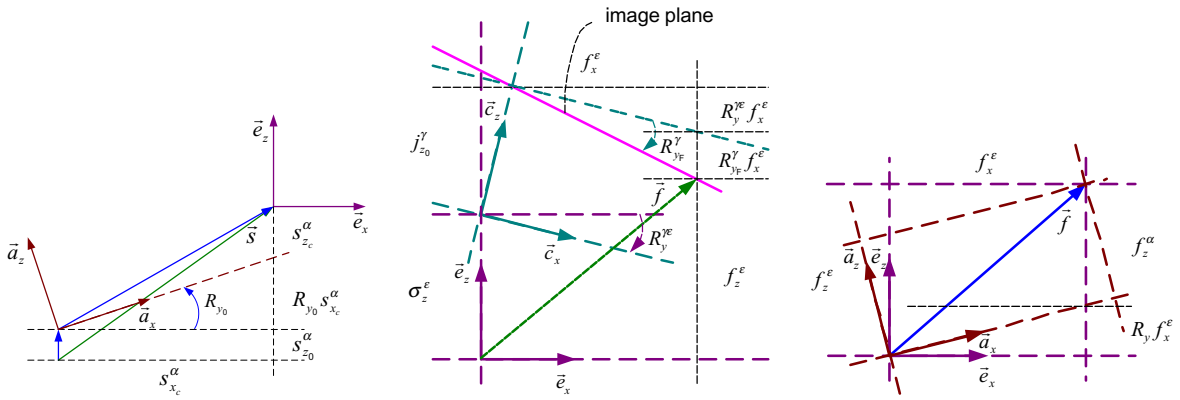
Now, the parameter vector contains parameter inaccuracies. The vector of known constants contains both known constant inaccuracies (all zero in this example) and sensor inaccuracies.

6.3.3 Reticle Stage Alignment inaccuracy relations

We have derived the linear parameter relations for the CS and IS measurements in the reticle stage alignment case from the vector relations depicted in Fig. 6.3. This derivation involves considering a zeroing error, a lens-to-stage offset, and the vector coordinates transformation as depicted in Fig. 6.5 for the XZ-plane. The machine design and initial situation at the start of the RSA sequence justifies that several geometric parameters can be considered as known constants. Furthermore, the derived coefficients are such that small deviations in them have a negligible effect on the parameters. Therefore, the obtained linear relations as formulated in (6.19) can also be used to relate parameter inaccuracies.

$$s^{\alpha}_{z_{\mathrm{c}}} + s^{\alpha}_{z_0} + R_{y_0} s^{\alpha}_{x_{\mathrm{c}}} = \beta_{\mathrm{CS}}. \tag{6.19a}$$

$$s^{\alpha}_{z_{\mathrm{c}}} + \sigma^{\varepsilon}_{z} - (R^{\gamma\varepsilon}_{y} - R_{y_0}) f^{\varepsilon}_{x} + s^{\alpha}_{z_0} + R_{y_0} s^{\alpha}_{x_{\mathrm{c}}} = \beta_{\mathrm{IS}}. \tag{6.19b}$$



(a) Zeroing offset (b) Lens-to-stage offset (c) Vector coordinates transformation

Figure 6.5: Derived views from the kinematic chains

For CS, relation (6.19a) involves a zeroing error only (Fig. 6.5(a)). For IS, relation (6.19b) involves both a zeroing error (Fig. 6.5(a)) and a lens-to-stage offset (Fig. 6.5(b)). As the lens-to-stage offset involves both the α and ε coordinate systems, relation (6.19b) also involves the vector coordinates transformation (Fig. 6.5(c)).

Measurements can be done at two $s^{\alpha}_{x_c}$ positions, a and b. Per $s^{\alpha}_{x_c}$ position, one CS sensor unit is available, each having its own $s^{\alpha}_{z_c}$ parameter. For $s^{\alpha}_{x_c}$ position a, four IS sensor units are available, whereas for position b two IS sensor units are available. Each sensor unit has its own f^{ε}_{x} coefficient, whereas each combination of measurement position and IS unit has its own $s^{\alpha}_{z_c}$ parameter. Therefore, Equation (6.19a) for CS can be expanded as follows:

$$s^{\alpha}_{z_{\mathrm{c,aCS}}} + s^{\alpha}_{z_0} + R_{y_0} s^{\alpha}_{x_{\mathrm{c,a}}} = \beta_{\mathrm{aCS}}, \tag{6.20a}$$

$$s^{\alpha}_{z_{\mathrm{c,bCS}}} + s^{\alpha}_{z_0} + R_{y_0} s^{\alpha}_{x_{\mathrm{c,b}}} = \beta_{\mathrm{bCS}}, \tag{6.20b}$$

whereas Equation (6.19b) for IS can be expanded as follows

$$s^{\alpha}_{z_{\mathrm{c,aIS}_i}} + \sigma^{\varepsilon}_{z} - (R^{\gamma\varepsilon}_{y} - R_{y_0}) f^{\varepsilon}_{x_{\mathrm{IS}_i}} + s^{\alpha}_{z_0} + R_{y_0} s_{x_{\mathrm{c,a}}} = \beta_{\mathrm{aIS}_i}, \qquad i = 1,2,3,4, \tag{6.21a}$$

$$s^{\alpha}_{z_{\mathrm{c,bIS}_i}} + \sigma^{\varepsilon}_{z} - (R^{\gamma\varepsilon}_{y} - R_{y_0}) f^{\varepsilon}_{x_{\mathrm{IS}_i}} + s^{\alpha}_{z_0} + R_{y_0} s_{x_{\mathrm{c,b}}} = \beta_{\mathrm{bIS}_i}, \qquad i = 1,2. \tag{6.21b}$$

Some examples of combined parameters are:

$$\tau^{\varepsilon}_{z} = \sigma^{\varepsilon}_{z} + s^{\alpha}_{z_0}, \tag{6.22a}$$

$$\nu^{\alpha}_{z} = s^{\alpha}_{z_0} + R_{y_0} s^{\alpha}_{x_{\mathrm{c}}}. \tag{6.22b}$$

These combined parameters can be substituted into (6.19) yielding the following equations:

$$\{\tau^{\varepsilon}_{z}\}: \qquad s^{\alpha}_{z_{\mathrm{c}}} + \tau^{\varepsilon}_{z} - (R^{\gamma\varepsilon}_{y} - R_{y_0}) f^{\varepsilon}_{x} + R_{y_0} s^{\alpha}_{x_{\mathrm{c}}} = \beta_{\mathrm{IS}}, \tag{6.23a}$$

$$\{\nu^{\alpha}_{z}\}: \qquad s^{\alpha}_{z_{\mathrm{c}}} + \sigma^{\varepsilon}_{z} - (R^{\gamma\varepsilon}_{y} - R_{y_0}) f^{\varepsilon}_{x} + \nu^{\alpha}_{z} = \beta_{\mathrm{IS}}. \tag{6.23b}$$

Note that these equations can also be instantiated for the different measurement positions.

6.4 Calibration sequencing

This section describes the constraints for instantiating a calibration task graph (instantiated, unselected TRS (2) in Fig. 6.1): the uninstantiated TRS (3) in Fig. 6.1. First, we consider parameter errors without variance. We define the system state and the TRS definition elements that parameterize the instantiation constraints, based on which we define the constraints afterwards. Then, we describe the consequences of adding variance to the parameter errors. Finally, we convert a sequence of calibration tasks into an instantiated unselected TRS as described in Section 6.2, thus incorporating parallelism possibilities.

6.4.1 Uninstantiated system definition

Let the elements in parameter vector $\mathbf{x}_{\text{pc}}$ be contained in set $\mathcal{X}$. Assuming that the parameter errors contain no variance, we define the state of a system with respect to parameter errors by $Sx : \mathcal{X} \rightarrow \mathbb{R} \cup \{\perp\}$, giving the error value of each parameter, which can also be undefined ($\perp$).

Whereas tasks in an instantiated TRS have a precedence relation, this is not the case for an uninstantiated TRS. Tasks can be instantiated from so-called meta-tasks. A computation meta-task involves deriving a certain -solvable sized- subset of parameters from a certain subset of equations from (6.8). Although there is no precedence relation between meta-tasks, constraints exist for instantiating meta-tasks ensuring that the resulting instantiated TRS definition is feasible: pre-conditions. A computation can be executed if all input parameters are defined. A measurement can only be executed successfully if the parameter inaccuracies are in capture range. These measurement capture range constraints can also be defined in matrix form:

$$\mathbf{C}\mathbf{x}_{\text{pc}} \leq \mathbf{d}. \tag{6.24}$$

Here, $\mathbf{d}$ contains capture ranges, and $\mathbf{C}$ contains coefficients that indicate which parameters are of importance for a certain measurement. Furthermore, execution of a meta-task results in changed error values of certain calibration parameters, which can be associated with post-conditions.

Example 4 (Capture range) *For a successful measurement, the point to be measured must be placed in capture range of the sensor. In the plane positioning example, the capture range expresses how close to the point to be measured the sensor needs to be positioned in order to capture the target. To be able to do so, we need a height estimate of the point to be measured that is accurate enough. Let d_p denote the capture range of a measurement of point p, i.e., the maximum allowed inaccuracy $s^{\alpha}_{z_p}$ of a height estimate of point p. We can define the following constraints for the two height measurements:*

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s^{\alpha}_{z_0} \\ R_y \\ R_x \\ R_{xy} \\ s^{\alpha}_{z_4} \\ s^{\alpha}_{z_5} \end{bmatrix} \leq \begin{bmatrix} d_4 \\ d_5 \end{bmatrix}. \tag{6.25}$$

The elements of an uninstantiated TRS (level 3 in Fig. 6.1) that define calibration meta-task pre- and post-conditions can be defined as follows:

- $\mathcal{T}_3$ is a finite set of uninstantiated calibration tasks, or meta-tasks. Two types of calibration meta-tasks exist: measurements and computations ($\mathcal{T}_3 = \mathcal{T}_{\text{m3}} \cup \mathcal{T}_{\text{c3}}$).

- Cb_3, Ce_3: $\mathcal{T}_3 \rightarrow \mathcal{P}(\mathcal{X}) \times \mathcal{P}(\mathbb{N})$ give the parameters involved in the pre-condition (input parameters) and in the post-condition (output parameters) of a meta-task. The meaning of the tuple elements depends on the type of meta-task and pre-condition or post-condition, and is given in the following table:

	Measurement: $t_3 \in \mathcal{T}_{\text{m3}}$	Computation: $t_3 \in \mathcal{T}_{\text{c3}}$
$Cb_3(t_3).0$	Input parameters	Input parameters
$Ce_3(t_3).0$	Output parameters	Output parameters
$Cb_3(t_3).1$	Equations of (6.24) used in pre-condition (rows of $\mathbf{C}$ and $\mathbf{d}$)	Equations of (6.17) used
$Ce_3(t_3).1$	Equations of (6.17) used in post-condition (rows of $\mathbf{A}_{\text{pcm}}$ and $\mathbf{b}_{\text{pcm}}$)	Equations of (6.17) used

Constraints on these definitions that follow from their meaning described in the previous section are:

1. For measurements, the equations of (6.24) identified by $Cb_3(t_{m3}).1$ involve exactly the input parameters: $Cb_3(t_{m3}).0$. This implies that the columns of $\mathbf{C}$ corresponding with the remaining parameters contain only zeros at the identified rows. On the other hand, the equations of (6.17) identified by $Ce_3(t_{m3}).1$ involve exactly the output parameters: $Ce_3(t_{m3}).0$. Hence, the columns of $\mathbf{A}_{\text{pcm}}$ corresponding with the remaining parameters contain only zeros at the identified rows.

2. For computations, $Cb_3(t_{c3}).1$ and $Ce_3(t_{c3}).1$ identify the same equations of (6.17) involving exactly the input and the output parameters: $Cb_3(t_{c3}).0 \cup Ce_3(t_{c3}).0$. The columns of $\mathbf{A}_{\text{pcm}}$ corresponding with the remaining parameters contain only zeros at the identified rows.

3. The number of output parameters of a meta-task $Ce_3(t_3).0$ may not exceed the number of solvable parameters given the equations of (6.17) used, identified by $Ce_3(t_3).1$.

6.4.2 Instantiating a calibration sequence

This subsection describes two functions: one to determine which meta-tasks are eligible to be instantiated in a certain system state, and one to update the system state for the execution of a meta-task. These functions can be used to verify a calibration sequence and can be embedded in an algorithm to automatically generate calibration sequences, similar to Chapter 5.

We say that $\mathbf{C}$ is of size $m \times n$, meaning that it has m rows and n columns. By c_{ij} we denote the element of $\mathbf{C}$ at the ith row and the jth column. By x_j we denote the element of $\mathbf{x}$ at the jth row, or $Sx(j)$. By d_i we denote the element of $\mathbf{d}$ at the ith row.

Let function *eligible*: $(\mathcal{X} \rightarrow \mathbb{R} \cup \{\perp\}) \rightarrow \mathcal{P}(\mathcal{T}_3)$ be a function that determines which meta-tasks are eligible to be instantiated in a certain system state (Sx):

$$eligible(Sx) = \{t | t \in \mathcal{T}_3, precond(t, Sx)\}. \tag{6.26}$$

Function *precond*: $\mathcal{T}_3 \times (\mathcal{X} \rightarrow \mathbb{R} \cup \{\perp\}) \rightarrow \mathbb{B}$ is a function that determines whether the pre-conditions of a meta-task are satisfied in some system state:

$$\begin{aligned} precond(t, Sx) = \ & (\forall j : j \in Cb_3(t).0 : Sx(j) \neq (\perp)) \\ & \wedge (t \in \mathcal{T}_{m3} \Longrightarrow (\forall i : i \in Cb_3(t).1 : \sum_{j=1}^{n} c_{ij} x_j \leq d_i)). \end{aligned} \tag{6.27}$$

We use $\mathbb{B}$ to denote the set of boolean values: $\mathbb{B} = \{\text{true}, \text{false}\}$. Function *precond* checks whether all input parameters are defined and in case of a measurement also whether the input parameter inaccuracies are in capture range.

To update the system state, we wish to express the values of the state variables that are updated as a function of the old state. To be able to do so, we introduce a composed vector $\bar{\mathbf{x}}$ and rewrite (6.17) into

$$\bar{\mathbf{A}}\bar{\mathbf{x}} = \mathbf{0}, \tag{6.28}$$

where

$$\bar{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_{\text{pcm}} & -\mathbf{I} \end{bmatrix}, \qquad \bar{\mathbf{x}} = \begin{bmatrix} \mathbf{x}_{\text{pc}} \\ \mathbf{b}_{\text{pcm}} \end{bmatrix}. \tag{6.29}$$

Let $\bar{\mathbf{x}}_t$ denote the inaccuracies of the parameters and knowns involved in meta-task t. For each meta-task t, a matrix $\mathbf{P}_t$ can be defined that extracts $\bar{\mathbf{x}}_t$ from $\bar{\mathbf{x}}$:

$$\bar{\mathbf{x}}_t = \mathbf{P}_t \bar{\mathbf{x}}. \tag{6.30}$$

Furthermore, a matrix $\mathbf{Q}_t$ can be defined that selects the relevant rows from $\mathbf{A}$. Using $\mathbf{Q}_t$ and $\mathbf{P}_t$, we can define a matrix $\bar{\mathbf{A}}_t$ that contains the coefficients for meta-task t as follows:

$$\bar{\mathbf{A}}_t = \mathbf{Q}_t \bar{\mathbf{A}} \mathbf{P}_t^{\mathrm{T}}. \tag{6.31}$$

Equations (6.30, 6.31) can be used to select the parameter relations involved in meta-task t from (6.28) as follows:

$$\mathbf{Q}_t \bar{\mathbf{A}} \mathbf{P}_t^{\mathrm{T}} \mathbf{P}_t \bar{\mathbf{x}} = \mathbf{0}. \tag{6.32}$$

We can separate the input and output parameter inaccuracies in (6.32) as follows:

$$\mathbf{Q}_t \bar{\mathbf{A}} \begin{bmatrix} \mathbf{P}_t^{\text{in}^{\mathrm{T}}} & \mathbf{P}_t^{\text{out}^{\mathrm{T}}} \end{bmatrix} \begin{bmatrix} \mathbf{P}_t^{\text{in}} \\ \mathbf{P}_t^{\text{out}} \end{bmatrix} \bar{\mathbf{x}} = \mathbf{0}. \tag{6.33}$$

Bringing the input parameter inaccuracies to the right-hand side yields:

$$\mathbf{Q}_t \bar{\mathbf{A}} \mathbf{P}_t^{\text{out}^{\mathrm{T}}} \mathbf{P}_t^{\text{out}} \bar{\mathbf{x}} = -\mathbf{Q}_t \bar{\mathbf{A}} \mathbf{P}_t^{\text{in}^{\mathrm{T}}} \mathbf{P}_t^{\text{in}} \bar{\mathbf{x}}. \tag{6.34}$$

Let $\bar{\mathbf{x}}_t^{\text{out}'}$ denote a solution for the output parameter inaccuracies after execution of meta-task t. A solution exists if the constraints for the uninstantiated TRS definition are satisfied. Using the pseudo-inverse ([4]) of $\mathbf{Q}_t \bar{\mathbf{A}} \mathbf{P}_t^{\text{out}^{\mathrm{T}}}$, denoted by $(\mathbf{Q}_t \bar{\mathbf{A}} \mathbf{P}_t^{\text{out}^{\mathrm{T}}})^+$, we find the following solution:

$$\bar{\mathbf{x}}_t^{\text{out}'} = \mathbf{P}_t^{\text{out}} \bar{\mathbf{x}} = -(\mathbf{Q}_t \bar{\mathbf{A}} \mathbf{P}_t^{\text{out}^{\mathrm{T}}})^+ \mathbf{Q}_t \bar{\mathbf{A}} \mathbf{P}_t^{\text{in}^{\mathrm{T}}} \mathbf{P}_t^{\text{in}} \bar{\mathbf{x}}. \tag{6.35}$$

When system (6.34) is fully determined, (6.35) provides an exact solution. In the case that (6.34) is overdetermined, an approximative least-squares solution is provided by (6.35). We can now express the inaccuracies of the updated parameters as a function of the old state as follows:

$$\bar{\mathbf{x}}_t^{\text{out}'} = \mathbf{U}_t^{\text{out}} \bar{\mathbf{x}}, \tag{6.36a}$$

where

$$\mathbf{U}_t^{\text{out}} = -(\mathbf{Q}_t \bar{\mathbf{A}} \mathbf{P}_t^{\text{out}^{\text{T}}})^{+} \mathbf{Q}_t \bar{\mathbf{A}} \mathbf{P}_t^{\text{in}^{\text{T}}} \mathbf{P}_t^{\text{in}}. \tag{6.36b}$$

Updating the system state for the execution of a meta-task t can be described in the form

$$\bar{\mathbf{x}}' = \mathbf{U}_t \bar{\mathbf{x}}. \tag{6.37}$$

Only the output parameters in $\bar{\mathbf{x}}$ need to be updated by the update matrix $\mathbf{U}_t$. We can construct $\mathbf{U}_t$ from an appropriately sized identity matrix and $\mathbf{U}_t^{\text{out}}$ by replacing the rows of the identity matrix corresponding with the output parameters by the corresponding rows of $\mathbf{U}_t^{\text{out}}$.

6.4.3 Adding parameter inaccuracy variance

To incorporate a calibration parameter distribution having a mean and variance component, we extend the state of a system such that $Sx : \mathcal{X} \to \mathbb{R} \cup \{\perp\} \times \mathbb{R}_0^+ \cup \{\perp\}$ gives the mean and variance of the inaccuracy of each parameter, which can also be undefined ($\perp$). Let $\bar{\mathbf{x}}^{\mu}$ denote the vector of mean values of parameter inaccuracies, and let $\bar{\mathbf{x}}^{\sigma^2}$ denote the vector of variance values of parameter inaccuracies.

To check whether the pre-condition of a measurement meta-task is satisfied, the maximum absolute value of the input parameters must be considered. This maximum absolute error can be derived from the mean and the variance of the parameter inaccuracies. Redefining function *eligible* : $(\mathcal{X} \to \mathbb{R} \cup \{\perp\} \times \mathbb{R}_0^+ \cup \{\perp\}) \to \mathcal{P}(\mathcal{T}_3)$ for the added variance concerns only redefining function *precond* : $\mathcal{T}_3 \times (\mathcal{X} \to \mathbb{R} \cup \{\perp\} \times \mathbb{R}_0^+ \cup \{\perp\}) \to \mathbb{B}$:

$$\begin{aligned} \mathit{precond}(t, Sx) = \ & (\forall j : j \in \mathit{Cb}_3(t).0 : Sx(j).0 \neq \perp \wedge Sx(j).1 \neq \perp) \\ & \wedge (t \in \mathcal{T}_{\text{m3}} \Longrightarrow (\forall i : i \in \mathit{Cb}_3(t).1 : \sum_{j=1}^{n} c_{ij} |x_j^{\mu} + c\sqrt{x_j^{\sigma^2}}| \leq d_i)). \end{aligned} \tag{6.38}$$

Scalar c can be chosen using *Chebychev's inequality*,

$$P(|x_j - x_j^{\mu}| \geq c\sqrt{x_j^{\sigma^2}}) \leq 1/c^2, \tag{6.39}$$

which implies that the probability that a random parameter differs from its mean by at least c standard deviations is less than or equal to $1/c^2$ [8].

From statistics, we know how to derive the mean and variance value $\mathbf{y}^{\mu}$ and $\mathbf{y}^{\sigma^2}$ of a linear combination $\mathbf{L}$ of variables $\mathbf{x}$, each having a mean and variance $\mathbf{x}^{\mu}$ and $\mathbf{x}^{\sigma^2}$:

$$\mathbf{y}^{\mu} = \mathbf{L}\mathbf{x}^{\mu}, \tag{6.40a}$$

$$\mathbf{y}^{\sigma^2} = \mathbf{L}^2 \mathbf{x}^{\sigma^2}. \tag{6.40b}$$

Here, $\mathbf{L}^2$ denotes a matrix containing the pointwise squared elements of $\mathbf{L}$.

However, the restriction for the derivation of variance values using (6.40) is that the input parameters are *independent* random parameters, which is not the case in general. In fact, a computation task *by definition* results in output parameters that depend on the input parameters. If these parameters are used together as inputs later in the sequence, (6.40) can not be used anymore. We assume that the initial parameter inaccuracies, i.e., the parameter inaccuracies defined before calibration starts, and the parameter inaccuracies resulting from measurements are independent. To be able to calculate the state transition from the current state, we have to log the dependencies of parameters $\bar{\mathbf{x}}$ somehow as a function of the independent parameters $\bar{\mathbf{x}}_\mathrm{i}$. Therefore, we define

$$\bar{\mathbf{x}} = \mathbf{L}_\mathrm{i}\bar{\mathbf{x}}_\mathrm{i}, \tag{6.41}$$

where $\mathbf{L}_\mathrm{i}$ is a linear transformation from the independent parameter vector $\bar{\mathbf{x}}_\mathrm{i}$ to current parameter vector $\bar{\mathbf{x}}$. We add $\mathbf{L}_\mathrm{i}$ to the system state.

Substituting (6.41) into (6.37) yields:

$$\bar{\mathbf{x}}' = \mathbf{U}_t\mathbf{L}_\mathrm{i}\bar{\mathbf{x}}_\mathrm{i}. \tag{6.42}$$

Because $\bar{\mathbf{x}}_\mathrm{i}$ contains only independent parameters, applying (6.40) to (6.42) yields:

$$\mathbf{x}^{\mu\prime} = \mathbf{U}_t\mathbf{L}_\mathrm{i}\mathbf{x}^{\mu}, \tag{6.43a}$$

$$\mathbf{x}^{\sigma^2\prime} = (\mathbf{U}_t\mathbf{L}_\mathrm{i})^2\mathbf{x}^{\sigma^2}. \tag{6.43b}$$

Hence, we can use the update matrix $\mathbf{U}_t$ to update $\mathbf{L}_\mathrm{i}$ for the effect of execution of meta-task t:

$$\mathbf{L}_\mathrm{i}' = \mathbf{U}_t\mathbf{L}_\mathrm{i}. \tag{6.44}$$

It appears that the inaccuracies can be derived from the inaccuracies of the initial independent parameters and knowns, $\bar{\mathbf{x}}_\mathrm{i}^0$. Therefore, we can replace $\bar{\mathbf{x}}$ by $\mathbf{L}_\mathrm{i}$ in the system state if we add $\bar{\mathbf{x}}_\mathrm{i}^0$ to the system definition.

Concluding, the initial system state can be defined by $\bar{\mathbf{x}}_\mathrm{i}^0$, and $\mathbf{L}_\mathrm{i}$ is suited to define the current state. Initially, $\mathbf{L}_\mathrm{i}^0$ equals $\mathbf{I}$. The following elements together parameterize the constraints that play a role in instantiating a calibration sequence (uninstantiated TRS):

- Equation (6.24): $\mathbf{C}\mathbf{x}_\mathrm{pc} \leq \mathbf{d}$, defining the capture ranges of measurements (pre-conditions).
- Equation (6.28): $\bar{\mathbf{A}}\bar{\mathbf{x}} = \mathbf{0}$, defining the linear relation between parameter inaccuracies, both for computations and measurements (post-conditions).
- $\mathcal{T}_3 = \mathcal{T}_\mathrm{m3} \cup \mathcal{T}_\mathrm{c3}$, defining the set of available meta-tasks.
- Cb_3, Ce_3: $\mathcal{T}_3 \to \mathcal{P}(\mathcal{X}) \times \mathcal{P}(\mathbb{N})$, defining the meta-task pre-conditions and post-conditions by associating them with the relevant parts of (6.24) and (6.28).

To be able to schedule the calibration sequence for timing analysis, the capabilities involved with resources need to be defined, denoted by $I_3 : \mathcal{T}_3 \to \mathcal{P}(\mathcal{C})$. This information is not used for instantiating a calibration sequence.

6.4.4 Conversion of a calibration sequence into an instantiated unselected TRS

To convert a calibration sequence cs into a level 2 TRS definition in the calibration domain, denoted by D_2^{cal}, we call the recursive function *convert* with (ε, cs) as argument. During recursion, the internal variable *mapcs* is used to record the mapping of the instantiated meta-tasks to tasks and to determine the task precedence relation.

$$convert(mapcs, cs) = \begin{cases} D_2^{\varepsilon} & \text{if } cs = \varepsilon \\ convert(mapcs +\!\!+ [(hd(cs), t_2)], tl(cs)) \cup D_2' & \text{if } cs \neq \varepsilon. \end{cases} \tag{6.45}$$

where D_2^{ε} denotes the empty TRS definition and ε denotes the empty sequence. The tuple (t_2, D_2') is specified by $addtask(mapcs, hd(cs))$.

Function $addtask : (\mathcal{T}_3 \times \mathcal{T}_2)^* \times \mathcal{T}_3 \to (\mathcal{T}_2 \times \mathcal{D}_2)$ instantiates a task t_2 for an instantiated meta-task t_3 to extend the level 2 TRS definition D_2. This is obtained by adding an additional task t_2 to the existing TRS definition D_2, copying the capabilities involved, and instantiating for each input parameter of t_2 a precedence edge from the previous task in the calibration sequence that had that parameter as an output parameter. Given a map sequence *mapcs*, $addtask(mapcs, t_3)$ specifies a tuple (t_2, D_2') such that

$$\begin{aligned} & t_2 \in \mathcal{T}_2' \wedge t_2 \notin \mathcal{T}_2 \\ \wedge \quad & (\forall c : c \in I_3(t_3) : c \in I_2'(t_2)) \\ \wedge \quad & (\forall x, xcs\text{: } x \in Cb_3(t_3).0, xcs = filterxcs(mapcs, x) \\ & \qquad : (hr(xcs).1, t_2) \in P_2' \\ &) \end{aligned} \tag{6.46}$$

Function *filterxcs* : $(\mathcal{T}_3 \times \mathcal{T}_2)^* \times \mathcal{X} \to (\mathcal{T}_3 \times \mathcal{T}_2)^*$ filters the tasks from a map sequence *mapcs* that have some parameter x as output parameter:

$$filterxcs(mapcs, x) = \begin{cases} \varepsilon & \text{if } mapcs = \varepsilon \\ filterxcs(\text{tl}(mapcs), x) & \text{if } mapcs \neq \varepsilon \wedge x \notin Ce_3(\text{hd}(mapcs).0).0 \\ [\text{hd}(mapcs)] +\!\!+ filterxcs(\text{tl}(mapcs), x) & \text{if } mapcs \neq \varepsilon \wedge x \in Ce_3(\text{hd}(mapcs).0).0. \end{cases} \tag{6.47}$$

6.5 Results

In this section, we analyse an example RSA sequence taken from a wafer scanner in industry [1]. The following sequence of levelling-related measurements is performed in this RSA sequence:

1. A CS measurement at position a: aCS.
2. An IS measurement at position a with medium accuracy: aIS-M.
3. An IS measurement at position a with fine accuracy: aIS-F.
4. A CS measurement at position b: bCS.
5. An IS measurement at position b with fine accuracy: bIS-F.

The capture ranges and inaccuracies for the different types of measurements are listed in Table 6.1. Several computations are performed in between the measurements. Table 6.2 presents the initial inaccuracy as well as the desired inaccuracy after calibration.

Table 6.1: Capture ranges and inaccuracies for the measurements in RSA.

Measurement	Parameter x_j	Capture $\lvert x_j^\mu + 3x_j^\sigma \rvert \leq$	Inaccuracy $3x_j^\sigma$	Unit 10^{-6}
CS	ν_z^α	5.0	0.06	[m]
IS Medium	$s_{z_c}^\alpha$	1.0	0.1	[m]
IS Fine	$s_{z_c}^\alpha$	0.25	0.025	[m]

Table 6.2: Initial, desired, and predicted inaccuracy of RSA.

Parameter x_j	Initial error $6x_j^\sigma$	Desired error $\lvert x_j^\mu + 3x_j^\sigma \rvert \leq$	Predicted error $\lvert x_j^\mu + 3x_j^\sigma \rvert$	Unit 10^{-6}
σ_z^ε	0.20	$\perp$	0.100	[m]
$s_{z_0}^\alpha$	1.5	$\perp$	0.042	[m]
$R_y^{\gamma\varepsilon}$	8.0	1.3	1.076	[rad]
R_{y_0}	4.0	0.1	0.093	[rad]
τ_z^ε	$\perp$	0.014	0.012	[m]

We have implemented the theory presented in the previous sections in MATLAB, and have simulated the RSA sequence using the theory described in the previous sections. Fig. 6.6 shows the trajectory of the predicted inaccuracies along the tasks in the RSA sequence. Every involved parameter or combined parameter is plotted on its own axis. The horizontal axis represents the tasks in the RSA sequence, whereas the vertical axes represent the parameter inaccuracies on a logarithmic scale, i.e., the predicted maximum parameter error ($c = 3$). The task numbers correspond with the order in which the meta-tasks are instantiated in the RSA. Only the top four parameters are defined at the beginning of the sequence ($\bar{\mathbf{x}}_\mathrm{i}^0$). The remaining parameters are computed along the sequence. The circular-shaped markers indicate the input parameters for each computation task, whereas the triangular-shaped markers indicate the output parameters for each calibration task.

The slope of the lines in Fig. 6.6 shows that measurements in a wafer scanner cause inaccuracy improvements of approximately one order of magnitude: tasks 3, 6, 11, 13, 18. Because of this, the computations can also change the inaccuracy with that order of magnitude. Most computations in the RSA sequence involve simple substitutions. However, computation tasks 14 through 16 demonstrate the dependencies between the different computations. Within these three tasks, two computation tasks (tasks 14 and 15) are involved that actually solve a linear system. Note that due to the sequential order of solving the linear systems, the computation sequence 14 through 16 predicts different inaccuracies than in the case that the involved parameter relations were solved at once. Update matrix $\mathbf{U}_t$ and logging matrix $\mathbf{L}_\mathrm{i}$ ensure that the sequential computations are performed in terms of independent random parameters only. Every calibration task in the sequence influences the inaccuracy of at least one parameter. Notice the redundancy of tasks 6 and 7, which is caused by the implementation of RSA in the wafer scanner.

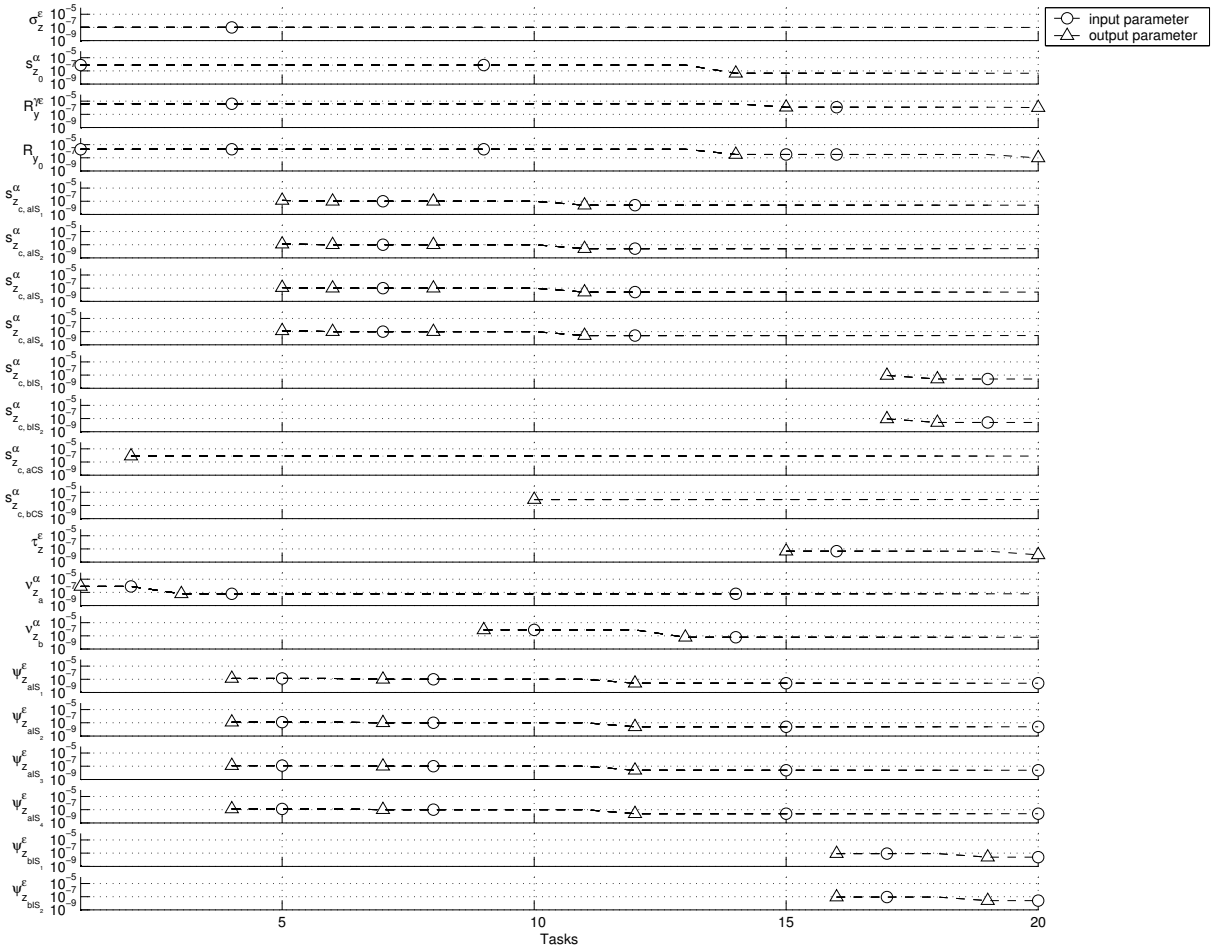


Figure 6.6: Predicted parameter inaccuracy along the RSA calibration sequence

We verify the instantiated RSA sequence by comparing the predicted inaccuracies before each measurement with the specified capture ranges, and comparing the predicted and desired end inaccuracies. Fig. 6.7 shows the ratio between the predicted maximum parameter error ($|x_j^{\mu} + 3x_j^{\sigma}|$) and the specified capture range for the measurements in the RSA sequence. From Fig. 6.7 we conclude that the predicted inaccuracies are all well in capture range. This raises the question whether or not a RSA sequence with less steps or shorter measurements is also possible. The predicted end inaccuracies are depicted in Table 6.2. From this table we can conclude that the desired end inaccuracy is reached.

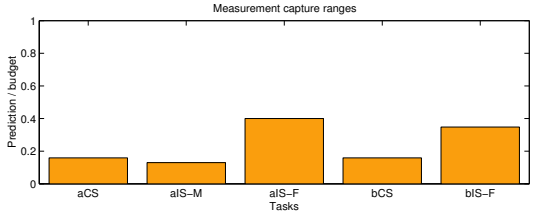


Figure 6.7: Capture range / actual inaccuracy

To be able to predict timing performance, we have converted the parameter dependencies between the tasks into a task precedence graph, which is displayed in Fig. 6.8. We used the theory described in Chapters 2 and 3 ([9, 10]) to generate two possible

schedules (see Fig. 6.9), both satisfying the task precedence graph. Note that besides the task precedence graph, additional information must be defined. The resources involved with measurements are modelled as resource rmeas, whereas computations are performed by resource rcomp. The alternative schedule leads to a shorter completion time than the schedule currently implemented due to more use of parallelism between measurement and computation tasks.

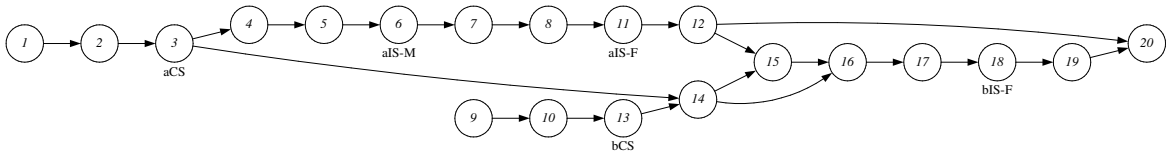


Figure 6.8: Task precedence graph corresponding with RSA

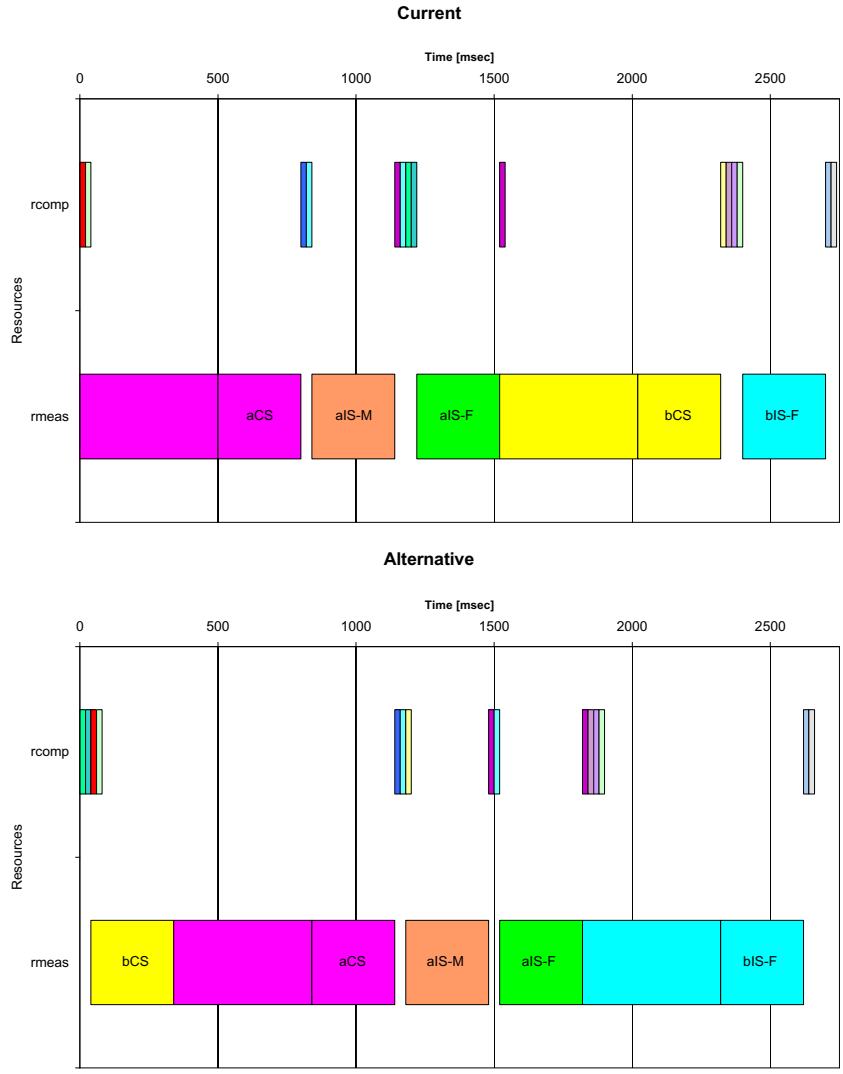


Figure 6.9: Current and alternative schedule for RSA

6.6 Conclusions

High-precision machines have to perform on-line calibrations to correct for imperfections in materials and drift in machine hardware. To be able to compensate for these imperfections with the required accuracy, calibration requires a sequence of calibration steps. Such a sequence involves measurements with multiple sensor types and computations to interrelate the different kinematic chains of the sensor types. This chapter describes the constraints to be taken into account in instantiating a calibration sequence. These constraints can be associated with a layered TRS framework that is suited for application in SMC.

We define the system state during calibration in terms of inaccuracy values of geometric parameters. We specify pre-conditions on the system state that have to be satisfied to successfully execute a calibration step. Furthermore, we specify post-conditions for execution of a calibration step by predicting the resulting state transition. These predictions follow from solving sets of linear equations relating the inaccuracy values of the geometric parameters. Inaccuracies are expressed by a mean and variance component to account for the random effects encountered in practice. We convert an instantiated calibration sequence into a task precedence graph (part of TRS definition level 2) using the parameter dependencies in the sequence. From that point, existing scheduling theory and tools concerning the lower part of the TRS framework can be used for timing performance analysis.

We illustrate the application of the theory by verifying an existing calibration sequence from a wafer scanner. Verification results confirm that the measurement input parameters are in capture range, and that the inaccuracy resulting after calibration suffices. Timing performance analysis shows that alternative schedules with different durations are possible. The described functionality can also be embedded in a search algorithm for automatic calibration sequence generation. If an effective search algorithm can be found, application of this theory in real-time SMC is possible. A remaining open issue for future research is the question which parameter combinations to choose [7, 15].

Acknowledgements

The authors would like to thank Wil Koenen for his useful suggestions, constructive criticism, and his help with the case.

References

[1] ASML, 2004. Information on wafer scanners available through URL `http://www.asml.com/`, item: products - lithography.

[2] Y.-J. Chiu and M.-H. Perng. Self-calibration of a general hexapod manipulator using cylinder constraints. *International Journal of Machine Tools & Manufacture*, 43:1051–1066, 2003.

[3] D. Daney, Y. Papegay, and A. Neumaier. Interval methods for certification of the kinematic calibration of parallel robots. In *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, April 2004.

[4] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, ML, 1983.

[5] J. M. Hollerbach. *A survey of kinematic calibration*. MIT Press, 1989.

[6] J. M. Hollerbach and C. W. Wampler. The calibration index and taxonomy for robot kinematic calibration methods. *International Journal of Robotics Research*, 15(6):573–591, 1996.

[7] S. Manetti and M. C. Piccirilli. A singular-value decomposition approach for ambiguity group determination in analog circuits. *IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications*, 50(4):477–487, 2003.

[8] D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, Inc., New York, 1994.

[9] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, N. C. W. M. Braspenning, and J. E. Rooda. Predictive scheduling in complex manufacturing machines: scheduling alternatives and algorithm. *submitted to IEEE TAC*.

[10] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, N. C. W. M. Braspenning, and J. E. Rooda. Predictive scheduling in complex manufacturing machines: machine-specific constraints. *submitted to IEEE TSM*.

[11] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, and J. E. Rooda. Design of supervisory machine control. In K. Glover and J. Maciejowski, editors, *Proceedings of the European Control Conference 2003*, 2003. CD-ROM.

[12] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 1995.

[13] J. Renders, E. Rossignol, M. Becquet, and R. Hanus. Kinematic calibration and geometrical parameter identification for robots. *IEEE Transactions on Robotics and Automation*, 7(6):721–732, 1991.

[14] J. Ryu and A. Rauf. A new method for fully autonomous calibration of parallel manipulators using a constraint link. In *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics Proceedings*, pages 8–12, 2001.

[15] J. A. Starzyk, J. Pang, S. Manetti, M. C. Piccirilli, and G. Fedi. Finding ambiguity groups in low testability analog circuits. *IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications*, 47(8):1125–1137, 2000.

[16] C. W. Wampler, J. M. Hollerbach, and T. Arai. An implicit loop method for kinematic calibration and its application to closed-chain mechanisms. *IEEE Transactions on Robotics and Automation*, 11(5):710–724, 1995.

[17] G. Yang and I. M. Chen. Kinematic calibration of modular reconfigurable robots. *Journal of Robotics Systems*, 16(4):213–225, 1999.

CHAPTER

SEVEN

MODEL CHECKER AIDED DESIGN OF A CONTROLLER FOR A WAFER SCANNER

This chapter is a slightly revised version of the paper *Model Checker Aided Design of a Controller for a Wafer Scanner* and is referred to in patent application ASML ref. P-1784.010.
The paper has been accepted for ISoLA 2004: 1st International Symposium on Leveraging Applications of Formal Methods, and an extended version has been submitted to IEEE Transactions on Automatic Control in July 2004.

Model Checker Aided Design of a Controller for a Wafer Scanner

Martijn Hendriks [1], Barend van den Nieuwelaar [2], Frits Vaandrager[1]

Abstract

For a case-study of a wafer scanner from the semiconductor industry it is shown how model checking techniques can be used to compute (i) a simple yet optimal deadlock avoidance policy, and (ii) an infinite schedule that optimizes throughput. Deadlock avoidance is studied based on a simple finite state model using SMV, and for throughput analysis a more detailed timed automaton model has been constructed and analyzed using the UPPAAL tool. The SMV and UPPAAL models are formally related through the notion of a stuttering bisimulation. The deadlock avoidance policy and the schedule that optimizes throughput were obtained within two weeks, which confirms once more that model checking techniques may help to improve the design process of realistic, industrial systems. Methodologically, the case study is interesting since two models (and in fact also two model checkers) were used to obtain results that could not have been obtained using only a single model (tool).

Keywords: Resource allocation systems, deadlock avoidance policy, throughput optimization, model checking, finite and timed automata, stuttering bisimulation.

7.1 Introduction

Scheduling and resource allocation problems occur in many different domains, for instance (1) scheduling of production lines in factories to optimize costs and delays, (2) scheduling of computer programs in (real-time) operating systems to meet deadline constraints, (3) scheduling of micro instructions inside a processor with a bounded number of registers and processing units, (4) scheduling of trains (or airplanes) over limited quantities of railway tracks and crossroads, and (5) mission planning for autonomous robots on spacecrafts. Typically, in each of these domain problems are solved using different approaches and mathematical tools. The EU IST project Ametist (`http://ametist.cs.utwente.nl/`) envisages a unifying framework for time-dependent behavior and dynamic resource allocation that crosses the boundaries of application domains.

In the Ametist approach, components of a system are modeled as *dynamical systems* with a state space and a well-defined dynamics. All that can happen in a system is expressed in terms of *behaviors* that can be generated by the dynamical systems; these constitute the semantics of the problem. Verification, optimization, synthesis and other design activities explore and modify system structure so that the resulting behaviors are correct, optimal, etc. Preferably, the limitations of currently known computational solutions should not influence modeling too much: only after the semantics of a problem is

[1] Nijmegen Institute for Computing and Information Sciences, University of Nijmegen, The Netherlands.

[2] Department of Mechanical Engineering, Eindhoven University of Technology, The Netherlands.

properly understood, abstractions and specialization due to computational considerations can intervene. In such situations, the soundness of abstractions should ideally also be proved, either via deductive verification or model checking.

The mission of Ametist is to extend this approach, which underlies the successful domain of *formal verification*, to resource allocation, scheduling and other time-related problems. The mathematical carrier for the Ametist methodology is the *timed automaton* model [2, 3], a modeling framework for discrete event dynamical systems that can handle quantitative timing delays between events. Some tools for *model checking* timed automata already exist, e.g., KRONOS [22] and UPPAAL [13]. Model checking is a method for formally verifying dynamical systems. Specifications about the system are expressed as temporal logic formulas, and efficient symbolic algorithms are used to traverse the model and to check (fully automatically) if the specification holds or not. We aim at further improving model checking tools for timed automata, investigating the applicability of these tools, and establishing links to tools developed in specific domains whenever appropriate.

In this chapter, as an illustration of the Ametist methodology, we use model checking techniques to solve the deadlock avoidance and throughput optimization problems for a realistic case of a wafer scanner from the semiconductor industry.

A major concern in the design of controllers for many resource allocation systems (RASs) is *deadlock*, a permanently blocking condition. There are three general ways of handling deadlock: (i) deadlock prevention, (ii) deadlock detection and resolution, and (iii) deadlock avoidance. Deadlock prevention restricts the system in such a way that deadlock is a priori impossible. As a consequence, performance may be unnecessarily low. Deadlock detection and resolution, on the other hand, is not restrictive at all and detects and resolves a deadlock at run-time. This, however, may be very expensive. Deadlock avoidance achieves a middle ground; it dynamically chooses the control actions to avoid the occurrence of deadlock. In this chapter, we show how a least restrictive deadlock avoidance policy (DAP) for the wafer scanner can be easily computed using SMV, a model checker for finite automata. This DAP can be represented by a very short predicate over the states of the wafer scanner, which can be used by the controller for the wafer scanner.

In addition, we use the timed automaton tool UPPAAL to define a refined model that adds timing constraints to address the issue of throughput optimization. We relate the UPPAAL model to the SMV model via the concept of *stuttering bisimulation* introduced by Browne, Clarke and Grumberg [5]. Since stuttering bisimulation preserves validity of CTL formulas (without nexttime operator), all properties (and in particular the DAP) that we established for the untimed model using SMV, carry over to the UPPAAL model. It is not possible to compute the least restrictive DAP directly for the UPPAAL model since (a) UPPAAL does not support full CTL, and (b) the state space of the UPPAAL model is so big that it cannot be fully explored. Using heuristics, however, we are able to use the UPPAAL model checker to find an infinite schedule that optimizes throughput.

Contribution. We obtained the deadlock avoidance policy and the schedule that optimizes throughput within two weeks, and we believe that our method can be applied by engineers with a background in computer science after training of only a few days. This confirms that model checking may help to improve the design process of realistic, industrial systems. Our DAP computation approach is referred to in a patent application of ASML [4], which shows its significance for industry. Methodologically, the case study is interesting since two models (and in fact also two model checkers) were used in com-

bination to obtain results that could not have been obtained using only a single model (tool). Our approach illustrates once more that building models that are just abstract enough for addressing a specific question, often provides a way to deal with the state space explosion problem. The SMV and UPPAAL models are formally related through the notion of a stuttering bisimulation. We are not aware of other work that addresses both deadlock avoidance and throughput optimization in (what essentially is) a single framework.

Related work. Other papers in which model checking tools are used to solve scheduling problems include a case study in which a control schedule for a smart card personalization system is synthesized using the SMV model checker [10], and a case study in which the UPPAAL model checker is used to find feasible schedules for a steel plant [9]. The present work is a follow-up on Chapter 3, which uses suboptimal deadlock avoidance heuristics to generate schedules that are not guaranteed to be optimal. The present work, however, gives a least restrictive (and thus optimal) DAP and a schedule that optimizes stationary throughput in the absence of errors.

Much research has been devoted to deadlock avoidance in RASs, see for instance [18, 19]. Discouraged by the NP-completeness of optimal deadlock avoidance for many RAS classes, see for instance [14], this kind of work generally focuses either on computation of suboptimal but polynomial DAPs or on optimal policies for very specific sub classes. Much of this work uses the Petri net formalism [17] for the modeling and analysis of RASs.

In [11] a deadlock free controller is constructed by an iterative process. The parallel composition of the controller and the plant is checked against deadlock by SMV. If a deadlock state is found, then the controller is adjusted to exclude the counterexample and the verification is run again. Otherwise, the controller is deadlock free. Finally, the work presented in [21] deals with verification of several DAPs using SMV.

Outline. First, Section 7.2 informally presents the case study. Section 7.3 then presents the SMV model and shows two ways of obtaining an optimal DAP using SMV. In Section 7.4, a UPPAAL model of the wafer scanner is proposed and infinite schedules which optimize throughput are computed. Finally, Section 7.5 draws some conclusions and gives directions for future work. A full version of this article, which includes all the proofs, is available as [12]. The complete SMV and UPPAAL models used in our case study are available at the URL `http://www.cs.ru.nl/ita/publications/papers/martijnh/`.

7.2 The EUV Machine

Lithographic machines, called *wafer scanners*, are used within the semiconductor industry to project chip designs on slices of silicon which are called wafers. A key performance characteristic of wafer scanners is throughput, i.e., the number of wafers that can be processed per time unit. For a typical recipe[1] it is desirable that the exposure operation (which uses the lens which is the most expensive part of the machine) is critical in optimal schedules. In order to maximize throughput, a controller should have a strategy that optimizes throughput in the absence of errors. Furthermore, we require that the controller is deadlock-free, since deadlock resolution is expensive.

Figure 7.1 schematically depicts a possible design of an *Extreme Ultra Violet machine* (EUV machine), which is a particular type of wafer scanner that is currently being devel-

[1]The timing parameters of the production depend on the chips to be produced.

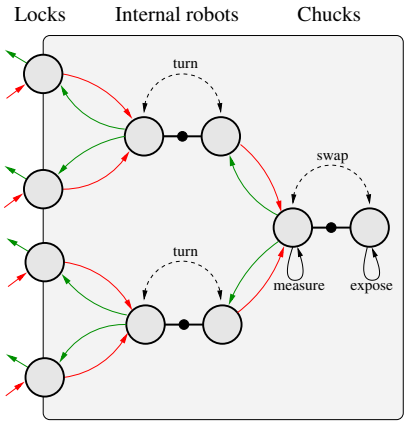


Figure 7.1: Wafer paths within the EUV machine.

oped by ASML. The inside of an EUV machine is kept vacuum as EUV light is absorbed by air. The wafer flow is presented in Figure 7.1.

First, the external track robot (which is not shown) puts a wafer in one of the four locks. This lock is depressurized, and then the wafer is picked up by one of the two internal robots. Each internal robot has two arms that can each hold a wafer and that are opposite to each other. The internal robot turns and puts the wafer on the closest chuck, which is in the so-called "measure position". The wafer is measured and a chuck swap is performed. The chuck with the measured wafer now is in the "expose position" and the wafer is exposed. After another chuck swap, the exposed wafer is picked up by one of the internal robots which turns and puts it in a depressurized lock. After the lock has been pressurized, the track robot removes the exposed wafer from the machine. Each wafer thus has a fixed recipe for its route: lock - internal robot - chuck - internal robot - lock. There is a choice which locks, internal robots and chucks are used by a wafer. An obvious question that arises is why we do not let the unexposed wafers flow through the upper two locks and let the exposed wafers exit through the lower two locks. In that case there are no crossing material paths which means that there is no deadlock possible by construction. The answer is twofold. First, if locks are unidirectional then filling the machine from the initial, empty, state takes unnecessarily long. Second, if locks are unidirectional then the depressurization operation might become critical instead of the exposure, since depressurization takes more than twice as long as exposure in a typical wafer recipe. As noted above, this is undesirable. In Section 7.4, we will prove that indeed the exposure subsystem is critical in the design of Figure 7.1, and that restricting the wafer flow to prevent deadlock a priori lowers both the throughput and the utilization of the exposure subsystem.

A typical example of a deadlock situation in the EUV machine would be a state in which all four robot arms hold unprocessed wafers, and both chucks hold processed wafers. A controller for the EUV machine should ensure that no such deadlock situation can ever be reached. The problem of finding such a control strategy is commonly referred to as the deadlock avoidance problem. The EUV machine is a disjunctive RAS according to the taxonomy of [15]. Instead of the traditional Petri net or graph based approaches to solving the deadlock avoidance problem, we will show in the next section how it can be tackled using the SMV model checker.

7.3 A Least Restrictive Deadlock Avoidance Policy

In this section, after a (very) brief introduction into SMV, we present our SMV model of the EUV machine, discuss how one can formalize the notion of deadlock as a temporal logic formula, and present the deadlock avoidance policy that we synthesized using SMV. The reader is referred to [7] and [16] for an extensive introduction into model checking and SMV.

7.3.1 SMV

In the approach supported by the SMV model checker, a system is modeled as a finite *transition system*, i.e. as a tuple $(S, s_{\text{init}}, \rightarrow)$ where S is a finite set of states, s_{init} is the initial state, and $\rightarrow \subseteq S \times S$ is the transition relation. We write $s \rightarrow s'$ instead of $(s, s') \in \rightarrow$. A state is defined as a valuation of a number of *state variables*. The value of state variable v in state s is denoted by $s(v)$. Furthermore, $s[v := c]$ denotes the state that is obtained by updating the value of v in state s to c. A *path* of a transition system is a sequence $s_0 s_1 s_2 \cdots$ such that for all i, $s_i \rightarrow s_{i+1}$. A state is *reachable* if it occurs on some path that starts in s_{init}.

In SMV, specifications are described in *Computation Tree Logic (CTL)*, a branching time temporal logic. Below some examples of CTL formulas are given, which should be sufficient to understand the present chapter. The basic building blocks of CTL are *atomic formula*, which denote functions from the set of states to $\{true, false\}$. For instance, if v is a state variable, then $v = 2$ is an atomic formula, which denotes the function from states to $\{true, false\}$ that maps a state s to *true* iff $s(v) = 2$. In this case, we say state s *satisfies* formula $v = 2$, notation $s \models (v = 2)$. Every atomic formula is a *state formula*. State formulas can be combined with Boolean connectives and *path operators*. We show three path operators that are relevant for this chapter. First, if ϕ is a state formula, then $\mathbf{AG}(\phi)$ also is a state formula. A state s satisfies $\mathbf{AG}(\phi)$, denoted by $s \models \mathbf{AG}(\phi)$, if for all paths $s_0 s_1 s_2 \ldots$ with $s = s_0$, and for all $i \geq 0$, $s_i \models \phi$. Second, if ϕ is a state formula, then $\mathbf{EF}(\phi)$ is also a state formula. We define $s \models \mathbf{EF}(\phi)$ if there exists a path $s_0 s_1 s_2 \ldots$ such that $s = s_0$ and $s_i \models \phi$, for some $i \geq 0$. Finally, if ϕ is a state formula, then $\mathbf{EG}(\phi)$ also is a state formula. We define $s \models \mathbf{EG}(\phi)$ if there exists a full path $s_0 s_1 s_2 \ldots$ with $s = s_0$ such that for all $i \geq 0$, $s_i \models \phi$.

7.3.2 An SMV Model of the EUV Machine

The EUV machine can be modeled conveniently and concisely in SMV. In fact, the full code is displayed in Figure 7.2.

For each of the 10 positions in the machine our model contains a state variable: an array `l` of size 4 for the locks, a 2-dimensional array `rb` of size 2×2 for the robots, and an array `c` of size 2 for the chucks. These state variables can either take value `e` (*empty*), which means that the position is empty, value `r` (*red*), which means that the position is occupied by an unexposed wafer, or `g` (*green*), which means that the position is occupied by an exposed wafer. Initially, the machine is completely empty and all state variables have value `e`.

To model the system dynamics, i.e., the movement and exposure of wafers, we introduce 22 asynchronous processes, which are executed in an interleaving fashion:

```
module main ()
{
  -- state variables
  l : array 0..3 of {e,r,g};
  rb: array 0..1 of array 0..1 of {e,r,g};
  c : array 0..1 of {e,r,g};

  -- initialization
  for (i=0; i<4; i=i+1)
    init(l[i]):=e;
  for (i=0; i<2; i=i+1)
    for (j=0; j<2; j=j+1)
      init(rb[i][j]):=e;
  for (i=0; i<2; i=i+1)
    init(c[i]):=e;

  -- system dynamics
  for (i=0; i<4; i=i+1)
    tl[i]: process entry_exit(l[i]);

  for (i=0; i<4; i=i+1)
    for (j=0; j<2; j=j+1)
      lr[i][j]: process move(l[i],rb[(i<2?0:1)][j]);

  for (i=0; i<2; i=i+1)
    for (j=0; j<2; j=j+1)
      for (k=0; k<2; k=k+1)
        rc[i][j][k]: process move(rb[i][j],c[k]);

  for (i=0; i<2; i=i+1)
    exp[i]: process expose(c[i]);

}
```

```
module entry_exit (p)
{
  if (p=e)
    next(p):=r;
  else if (p=g)
    next(p):=e;
}

module move (lft,rgt)
{
  if (lft=r && rgt=e)
  {
     next(lft):=e;
     next(rgt):=r;
  }
  else if (lft=e && rgt=g)
  {
     next(lft):=g;
     next(rgt):=e;
  }
}

module expose (p)
{
  if (p=r)
     next(p):=g;
}
```

Figure 7.2: SMV model of EUV machine.

- For each of the 4 locks `i` we have process `tl`[`i`], which may either put an unexposed wafer in lock `i` if it is empty, or move an exposed wafer from the lock to the track robot. In the definition of process `tl`[`i`] we use an auxiliary function `entry_exit` that describes the state change that results from running this process.
- For each of the 16 pairs of positions `i`, `j` such that `i` is on the left of `j` and a wafer can move directly from `i` to `j` (or back), we introduce a process that takes care of moving unexposed wafers from `i` to `j`, and exposed wafers from `j` back to `i`. In the definition of these processes we use a function `move`(`lft`, `rgt`) that describes the state change that results from moving a wafer from `lft` to `rgt` or vice versa.
- For each of the 2 chucks `i` we introduce a process `exp`[`i`] that models exposure of the wafer. An auxiliary function `expose` describes the state change that results from exposing the wafer at position `p`: the value of the corresponding state variable changes color from `r` (red) to `g` (green).

In the SMV model we abstract from the turning of internal robots. So a wafer can

be picked up by both arms of an internal robot (possibly, the robot first has to turn). Similarly, the SMV model abstracts from chuck swaps and the measure operation. In Section 7.4, we present a more detailed model of the EUV machine in which we do not abstract from these aspects.

As it turns out, our SMV model has 57116 reachable states, which is close to the total number of states which equals $3^{10} = 59049$. An example of an unreachable state is one in which the machine is completely filled with exposed wafers. Transition systems of this size can very easily be handled by SMV and the computer hardware that is available today. In fact, SMV routinely handles systems with 10^{20} states and beyond, so we expect that our approach can also be applied to considerably larger designs.

7.3.3 Defining Deadlock and Safety in SMV

Standard textbooks on operating systems, e.g. [20], state four conditions for deadlock in systems that consist of *processes* that compete for *resources*. The first three conditions concern the model itself and are necessary, and the fourth condition concerns the states of the model and is necessary and sufficient when the first three are met: (i) mutual exclusion: only one process may use a resource at a time, (ii) hold and wait: a process may hold allocated resources while awaiting assignment of others, (iii) no preemption: no resource can be forcibly removed from a process that is holding it, and (iv) circular wait: a closed chain of processes exists such that each process holds at least one resource needed by the next resource in the chain.

In the EUV machine, the wafers are modeled as the processes and they compete for the positions in the machine that constitute the resources. The model of the EUV machine satisfies the first three conditions for deadlock. The fourth condition, which thus is necessary and sufficient for deadlock, can be formalized with help from a *needs* function, that specifies for each wafer the set of positions it may move to. Let P denote the set of positions in the EUV machine. For $p \in P$ and $c \in \{\mathtt{r}, \mathtt{g}\}$, we define $\mathit{needs}(p, c) \subseteq P$ to be the set of positions (different from p) to which a wafer with color c at position p may move next. In particular, if p is a chuck, then $\mathit{needs}(p, \mathtt{r}) = \mathit{needs}(p, \mathtt{g}) = R$, where R is the set of positions of the internal robots. If s is a state and p a position then we use $\mathit{needs}^s(p)$ as an abbreviation for $\mathit{needs}(p, s(p))$. The circular wait property can now be defined as follows.

Definition 7.3.1 (Circular wait) *A state s has a* circular wait *in $Q \subseteq P$ iff $s(q) \neq \mathtt{e} \wedge \emptyset \neq \mathit{needs}^s(q) \subseteq Q \neq \emptyset$ for all $q \in Q$.*

It is not possible to directly formulate the circular wait property in terms of CTL, so some encoding is required. The basic idea is that the machine has a circular wait in a subset Q of positions iff the wafers in Q will never be able to move again. Observe that if in our model a transition $s \to s'$ moves a wafer from place p to place p', then p is empty in s'. Thus, the property that some wafer cannot move anymore can be formalized in CTL as follows.

Definition 7.3.2 (Jam) *A position p is* jammed *in state s iff $s \models \mathbf{AG}(p \neq \mathtt{e})$. A state s is* jammed *iff some position is jammed in s.*

Proposition 7.3.3 below asserts the equivalence of the circular wait and jammed properties, thereby providing us with a way to express deadlocks in the model of the EUV

machine in CTL. It has only been proven for our model of the EUV machine, but from the proofs it should be clear that these results can be generalized to a whole class of resource allocation problems.

Proposition 7.3.3 *A state has a circular wait in some Q iff it is jammed.*

In the remainder of this chapter, we will say that a state is *deadlocked* if it has circular wait, i.e., if it is jammed. The question that we need to answer is whether and how we can prevent the system of entering a deadlocked state. In Dijkstra's paper on the *banker's algorithm* [8], the first published deadlock avoidance algorithm, a state is defined to be *safe* if "all processes can be run to completion". In our case, the wafers are the processes and "a wafer is run to completion" if it exits the machine. Thus, Dijkstra's definition can be translated to CTL as follows.

Definition 7.3.4 (Safe states) *A state s is* safe *iff* $s \models \mathbf{EF}\left(\bigwedge_{p \in P}(p = \mathsf{e})\right)$.

Note that in general safe and not being deadlocked are different things. If a state s is not deadlocked then $s \models \bigwedge_{p \in P} \mathbf{EF}(p = \mathsf{e})$, i.e., each individual position can be emptied, but it need not be the case that all positions can be emptied simultaneously. If a state is deadlocked it is unsafe, but if it is unsafe it need not be deadlocked. However, in many cases and (according to SMV) in particular for our model of the EUV machine, the following property does hold[2]:

$$\mathbf{AG}(\text{safe} \iff (\mathbf{EG}\ \neg\text{deadlock})). \tag{7.1}$$

This formula suggests a simple least restrictive DAP: just keep the system in a safe state. This policy can be realized for the EUV machine. Every non-initial safe state has at least one safe successor (different from itself). In addition, we verified using SMV that all successors of the initial state are again safe, which completes the proof that the system can be kept in a safe state.

7.3.4 A Least Restrictive DAP

In order to actually build a controller that always keeps the system in a safe state, it would clearly be very helpful to have a simple, yet exact characterization of the set of safe states. We see two ways to obtain such a characterization.

1. When checking whether the initial state is safe, SMV computes a *binary decision diagram* (BDD, see [6]) which provides a compact representation of the set of safe states. With the available SMV releases it is not possible to get the BDD out. However, since there is an open-source distribution available solving this problem should just be a matter of programming.

[2] In fact, in the EUV machine a state is safe if and only if it has no deadlock. It is easy to come up with variations of the machine with states that are not safe and not deadlocked, for example a design in which the internal robots only have one arm. In such cases, in order to make formula (7.1) hold, we need to require weak fairness for all processes in the SMV model to exclude runs in which no progress is made due to infinite stuttering of some components.

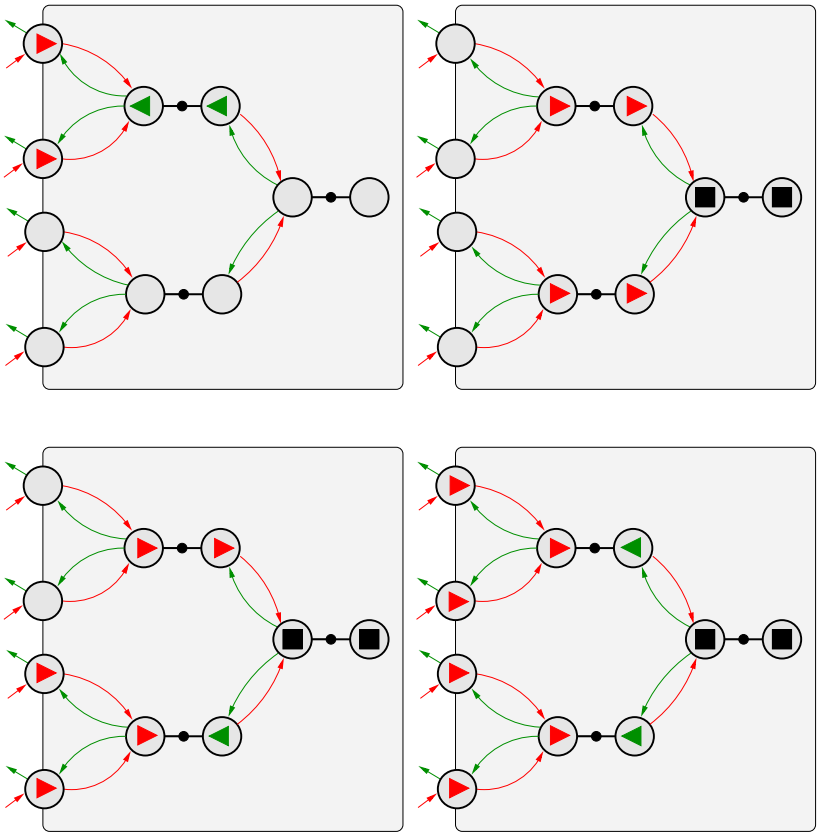


Figure 7.3: The four unsafe scenarios (modulo symmetry) in the EUV machine.

2. The set of safe states can be manually characterized by the following iterative procedure:

$$
\begin{array}{l}
S := \mathit{true} \\
\textbf{while} \neg\,(s_{\text{init}} \models \textbf{AG}(\text{safe} \iff S)) \\
\qquad S := S \wedge (\neg C)
\end{array}
$$

   where C is the characterization of the last state of the counter example that is generated by SMV.

The first approach enables a least restrictive DAP with linear time complexity, since checking whether a state is included in a BDD takes $\mathcal{O}(n)$ operations, where n is the number of booleans from which the BDD is composed (20 in case of the EUV machine). The size of the BDD, however, can in the worst case be exponential in the number of booleans. A second drawback is that it can be difficult to derive individual unsafe and/or deadlock situations from a BDD, which may be required during the design phase of the system. The second approach can quickly become practically infeasible since all unsafe states are explicitly enumerated. If it is carried out manually, however, then it might be possible to abstract from irrelevant state information and to visualize the various unsafe situations in the system. Of course, this requires some effort and creativity from the analyst. The second approach has been used to characterize the safe states of the EUV machine. With five iterations, we found four unsafe situations, depicted in Figure 7.3, which happen to characterize all deadlocks. A right-pointing arrow represents an unexposed wafer, a left-pointing arrow represents an exposed wafer, and a black square

represents an unexposed or exposed wafer. The predicate S that exactly characterizes the set of safe states is the negation of the situations shown in Figure 7.3, and can be described in the input language of SMV with 695 characters.

Note that SMV can also be used to obtain a simple under-approximation of the set of safe states (when, e.g., the BDD is too large to use and the iterative process is too time consuming). If C is a candidate for a simple under-approximation, then this can be verified with the CTL property $\mathbf{AG}(C \Rightarrow \text{safe})$. Again, counter-examples can be used to correct C while retaining low complexity. Note, however, that it now becomes necessary to ensure that the initial state is reachable from any state in C (this is true by definition for the set of all safe states).

7.4 Throughput Analysis

A first objective for a controller of the EUV machine is to avoid deadlocks. In the previous section, using our SMV model, we synthesized a least restrictive control policy that achieves this. A second key objective for a controller of the machine of course is to maximize throughput. Our SMV model is not sufficiently detailed to address this issue since, for instance, relevant information about the delays in the locks and the speed of the robots has not been included. Also, the SMV model abstracts from the delays due to turning of the internal robots, measuring of wafers, and swapping of the chucks. Therefore, in this section, we present a more refined *timed automata model* ([2, 3]), which contains sufficient information to address the throughput issue.

In order to define and analyze our model, we used the UPPAAL model checking tool. UPPAAL supports modeling of systems in terms of networks of timed automata which are extended by blocking synchronization and bounded integer variables. Similarly to SMV, the semantics of a UPPAAL model is defined by a transition system. In addition to the discrete part, the states also contain a real-valued clock valuation. For these models, the UPPAAL model checker can decide a subset of *Timed Computation Tree Logic* (TCTL, see [1]). For a detailed account of UPPAAL we refer to [13] and to `http://www.uppaal.com`.

After presenting the UPPAAL model of the EUV machine in Section 7.4.1, we discuss the relationship between the UPPAAL and SMV models in Section 7.4.2. Then, in Section 7.4.3, we use UPPAAL to derive a schedule for the EUV machine that optimizes throughput.

7.4.1 UPPAAL Model

The UPPAAL model of the EUV machine contains the same state variables as the SMV model for the positions in the machine: arrays `l`, `rb` and c, which may take the same values `e`, `r` and `g` to indicate that a position is respectively empty, filled with an unexposed wafer, or with an exposed wafer. In addition, the UPPAAL model has a number of Boolean state variables to ensure "physical integrity". For instance, an internal robot can only access a lock if it is vacuum. This requirement is modeled using the Boolean `lb`[`id`] for lock number `id`. The model consists of 12 automata, of which 11 model physical components of the machine: the track robot, the four locks, the four robot arms (two for each of the robots), and the two chucks. These automata move wafers around with certain delays and according to the material paths as specified in Section 7.2. An additional automaton, the *observer*, is used for throughput optimization.

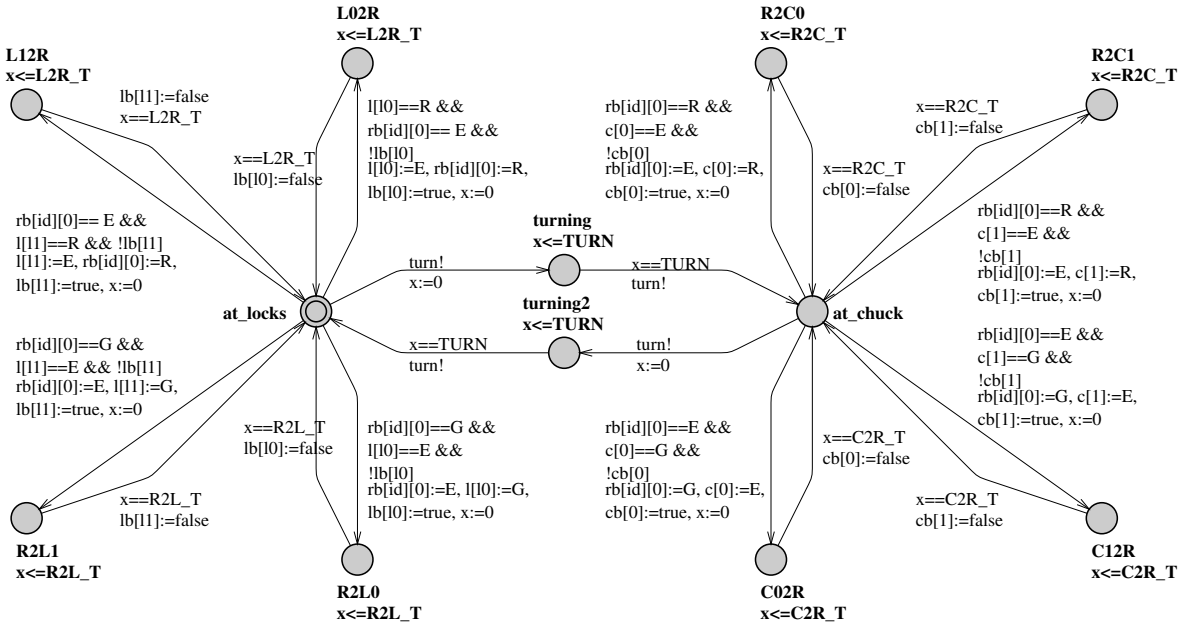


Figure 7.4: Template for a robot arm.

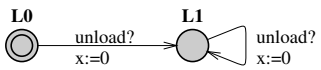


Figure 7.5: Process for the observer.

To illustrate the modeling in UPPAAL, we present the template for one arm of an internal robot, see Figure 7.4. This template has four parameters: a constant *id* that identifies the internal robot to which the arm belongs, two constants *l0* and *l1* that identify the locks to which the robot arm has access, and a channel *turn*. When a robot arm is at the locks, then it can get a wafer from a lock (*L02R* and *L12R*), or it can put a wafer in a lock (*R2L0* and *R2L1*). Of course, it can only perform these actions if the lock is vacuum, and if the wafer flow is as specified in Section 7.2. Similarly, when a robot arm is at the chucks then it can load/unload a wafer to/from the chuck that is at the *measure* location. The *cb* variables are used to ensure that only one robot arm has access to the chuck at a time and that the chuck cannot execute a transition while the robot arm is loading/unloading a wafer.

Figure 7.5 shows the observer process which, as we will explain in more detail in Section 7.4.3, is used to ensure progress in the model. This process measures the time until the first wafer exits the system (this is called an unload event) in location *L0*, and the time between two consecutive unload events in location *L1* using its local clock x.

7.4.2 Bisimulation between SMV and UPPAAL models

Clearly, there is a relationship between the SMV model and the UPPAAL model. The SMV model is an abstraction from the UPPAAL model, which has the property that every transition in the UPPAAL model can be simulated in the SMV model, and vice versa. Formally, the relationship between the two models can be expressed as a *stuttering bisimulation* relation in the sense of [5]. Stuttering bisimulations are defined in terms of

Kripke structure, an extension of transition systems in which to each state a set of atomic propositions is associated that hold in that state.

For a proof sketch using Kripke structures and stuttering bisimulation we refer to [12]. From that we may conclude that all the results on deadlock avoidance established using SMV in Section 7.3 carry over to the UPPAAL model. It is not possible to obtain these results directly using the UPPAAL tool since (a) UPPAAL does not support full CTL, and (b) the state space of the UPPAAL model is so big that it cannot be fully explored.

7.4.3 Finding an Optimal Schedule

As mentioned above, the *observer* process of Figure 7.5 observes unload events. It starts in location *L0* and upon the first unload event it resets its local clock x and enters location *L1*. In location *L1* the clock is reset whenever an unload event takes place. The observer is used to find an infinite schedule that takes at most H time units until the first unload event, and that has at most S time units between two unload events. Such a schedule is specified by the following TCTL property that can be checked by UPPAAL.

$$\mathbf{EG}((observer.L0 \Rightarrow observer.x \leq H) \wedge (observer.L1 \Rightarrow observer.x \leq S)) \qquad (7.2)$$

If this property is satisfied, then UPPAAL can return an example execution that consists of a path followed by a cycle. Such an execution thus gives an infinite control schedule for the wafer scanner with a *stationary* throughput of at least one wafer per S time units. Unfortunately, the size of the reachable state space prevents UPPAAL from finding such an execution directly. We therefore added heuristics to the model to prune the state space:

1. The DAP derived in the previous section has been used to avoid unsafe material configurations of the machine.

2. Some transitions are useless (or suboptimal) in certain states, e.g., an internal robot can always turn, but this is useless if it does not hold wafers. The state space has been reduced by adding guards that prevent such useless behavior.

3. The optimal behavior of the locks in the initial phase (the filling of the machine) differs from their optimal behavior in the stationary phase. Therefore a heuristic has been added to enforce this difference: a lock can pressurize when it contains either an exposed wafer, or it is empty and the machine is not yet filled with enough wafers to be in the stationary state.

4. Some transitions have been made *urgent* (greedy): they must be taken as soon as they are enabled. For instance, if the DAP allows loading a wafer to a lock, then this must be done immediately.

Note that using urgent transitions without the DAP may be an unwise idea, since this can result in many deadlocks with the effect that an execution satisfying Property 7.2 does not exist anymore in the model. Also note that at least the last three heuristics may remove good schedules.

A lower bound on the time until the first unload event, min_h, can easily be derived from the model. It is also easy to see that the minimal separation time between exposed wafers that appear at the chuck that is in the measure position (and can therefore be

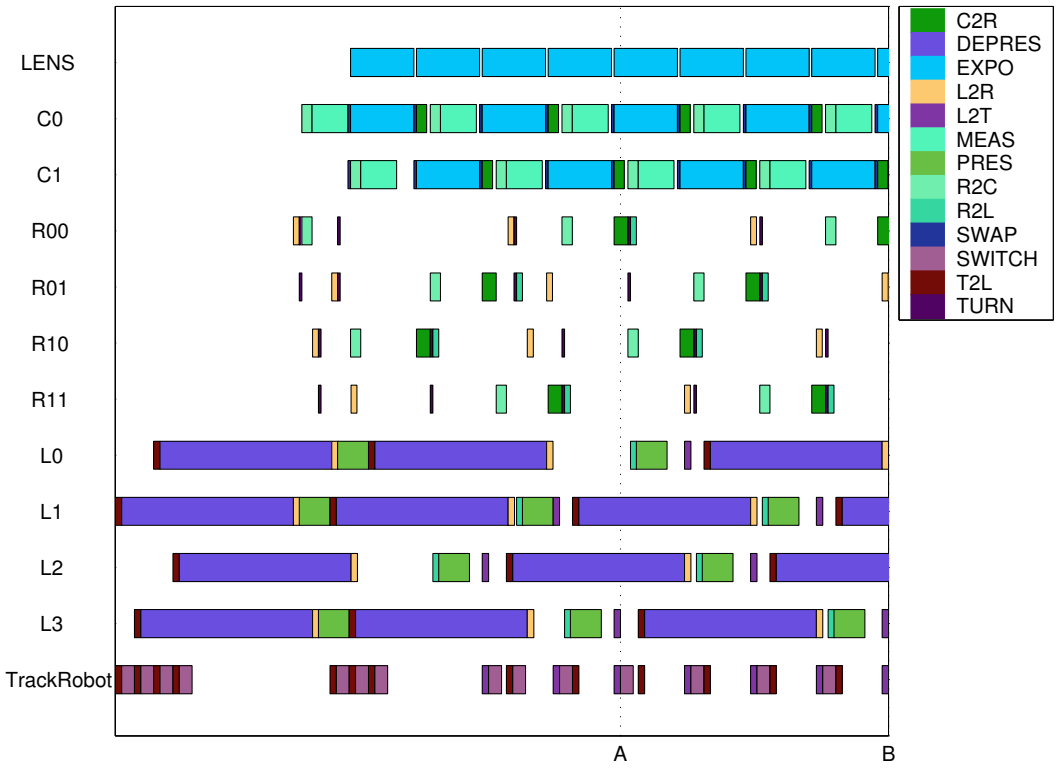


Figure 7.6: A schedule that optimizes the stationary throughput of the EUV machine. The cyclic part of the schedule consists of the interval between points A and B. Note that the operation of the lens is only interrupted by the chuck swap (which is necessary).

picked up by an internal robot) equals $min_s = EXPOSE + SWAP$, where the former is the time needed for the expose operation and the latter is the time needed for the chuck swap. Therefore, the theoretical maximal stationary throughput of the machine is at most one wafer per min_s time units. For the UPPAAL model with heuristics it is possible to find an execution that satisfies Property 7.2 for a value of H that is 5% larger than min_h and for $S = min_s$. Figure 7.6 shows this schedule that optimizes the stationary throughput of the EUV machine.

It took only little effort to change the UPPAAL model in order to analyze two alternative machine designs w.r.t. throughput. In the first design, the incoming wafers have been restricted to the upper two locks and the outgoing wafers to the lower two locks (to prevent deadlock a priori; see Section 7.2). We can easily find an optimal schedule with $S = 1.61 \cdot min_s$ that shows that not the expose operation but the locks have become critical. This confirms our suspicion that has been stated in Section 7.2. The second alternative design consists of only two locks and one internal robot. We can easily find a schedule with $S = 1.82 \cdot min_s$, but we cannot guarantee that this is an optimal schedule.

7.5 Conclusions

The SMV model checker has been used successfully to characterize the set of safe states of the EUV machine. This characterization consists of a very short boolean expression

over the places in the machine and is useful for the design of an actual controller since deadlock can easily be avoided by examining the possible successor states of the current state. Since the characterization is exact, the controller implements a least restrictive (optimal) deadlock avoidance policy. The approach is suited for industrial practice as no expertise on mathematical proofs is needed. Furthermore, we used the UPPAAL model checker to compute infinite schedules for the EUV machine that optimize stationary throughput. It took little effort to change the UPPAAL model in order to analyze two alternative machine designs.

In theory, our approach can be applied to a broad class of resource allocation systems. As always when using model checking, the state space explosion is the main problem for scalability. Building models that are just abstract enough for addressing a specific question, often provides a good way to deal with the state space explosion problem. Furthermore, in both cases, to obtain the results presented manual interventions are needed: to characterize the set of safe states in SMV and to define the scheduling heuristics in UPPAAL. Nevertheless, in our view, the present work nicely illustrates the usefulness of model checking techniques to support the design process of applications that involve resource allocation and scheduling.

Acknowledgements. The authors thank Biniam Gebremichael for his useful suggestions concerning the SMV model, and the anonymous reviewers for their helpful comments on a preliminary version of the present paper.

References

[1] R. Alur, C. Courcoubetis, and D. L. Dill. Model checking in dense real time. *Information and Computation*, 104:2–34, 1993.

[2] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *17th International Colloquium on Automata, Languages and Programming*, pages 322–335, 1990.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[4] ASML, 2004. Information on wafer scanners available through URL `http://www.asml.com/`, item: products - lithography.

[5] M.C. Browne, E.M. Clarke, and O. Grümberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1,2):115–131, 1988.

[6] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, C-35(8):677–691, August 1986.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.

[8] E. W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages: NATO Advanced Study Institute*, pages 43–112. Academic Press, 1968.

[9] A. Fehnker. Scheduling a steel plant with timed automata. In *Proceedings of the sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*. IEEE Computer Society Press, 1999.

[10] B. Gebremichael and F. W. Vaandrager. Control synthesis for a smart card personalization system using symbolic model checking. In K. G. Larsen and P. Niebert, editors, *Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, number 2791 in LNCS, pages 189–203. Springer–Verlag, 2004.

[11] V. Hartonas-Garmhausen, E. M. Clarke, and S. Campos. Deadlock prevention in flexible manufacturing systems using symbolic model checking. In *IEEE Conference on Robotics and Automation*, volume 1, pages 527–532, 1996.

[12] M. Hendriks, N. J. M. van den Nieuwelaar, and F. W. Vaandrager. Model checker aided design of a controller for a wafer scanner. Report NIII-R0430, Nijmegen Institute for Computing and Information Sciences, University of Nijmegen, The Netherlands, June 2004.

[13] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1/2):134–152, 1997.

[14] M. Lawley and S. A. Reveliotis. Deadlock avoidance for sequential resource allocation systems: Hard and easy cases. *International Journal of Flexible Manufacturing Systems*, 13(4):385–404, 2001.

[15] M. Lawley, S. A. Reveliotis, and P. Ferreira. Design guidelines for deadlock handling strategies in flexible manufacturing systems. *International Journal of Flexible Manufacturing Systems*, 9(1):5–30, January 1997.

[16] K. L. McMillan. *Symbolic Model Checking*. PhD thesis, Carnegie Mellon University, Pittsburgh, May 1992.

[17] T. Murata. Petri nets: Properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[18] J. Park and S. A. Reveliotis. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. *IEEE Transactions on Automatic Control*, 46(10):1572–1583, 2001.

[19] S. A. Reveliotis, M. Lawley, and P. Ferreira. Polynomial-complexity deadlock avoidance policies for sequential resource allocation systems. *IEEE Transactions on Automatic Control*, 42(10):1344–1357, 1997.

[20] W. Stallings. *Operating Systems – Internals and Design Principles*. Prentice–Hall, 1998.

[21] Y. Wang and Z. Wu. Deadlock avoidance control synthesis in manufacturing systems using model checking. In *IEEE American Control Conference*, volume 2, pages 1702–1704, 2003.

[22] S. Yovine. KRONOS: a verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(1/2):123–133, 1997.

CHAPTER

EIGHT

A DEDICATED SCHEDULING VERIFICATION APPROACH

This chapter is based on the paper *A Dedicated Verification Approach for Scheduling in Complex Manufacturing Machines* (TU/e Computer Science Reports 04/27), and defines some of the prerequisites for formal verification of the control concept proposed in this thesis. The paper has been protected in patent application ASML ref. P-1784.010, and will be submitted to the European Control Conference 2005.

A Dedicated Verification Approach for Scheduling in Complex Manufacturing Machines

N.J.M. van den Nieuwelaar[12], M.M.H. Driessen[3], J.F. Groote[3]

[1]ASML Netherlands B.V., De Run 6501, 5504 DR Veldhoven, The Netherlands.

[2]Department of Mechanical Engineering, Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

[3]Division of Computer Science, Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

N.J.M.v.d.Nieuwelaar@tue.nl, M.M.H.Driessen@student.tue.nl, J.F.Groote@tue.nl

Abstract

Dynamic scheduling of the production in complex industrial manufacturing machines can lead to problems such as, for instance, deadlocks bringing the production to a standstill. An approach to ensure that deadlock is avoided in these highly flexible systems is to investigate all potential schedules. But verification of industrially sized systems by state space traversal is practically impossible due to combinatoric effects causing state-space explosion. We present an effective approach to overcome this problem. The approach exploits specific characteristics of the systems and properties under consideration. Three situations in which state-space reduction techniques can be applied have been identified, formally defined and implemented in a dedicated checker. Results show that two of the techniques can reduce state spaces with an exponential factor, and one technique has a linear effect. The application on a number of wafer scanners shows that this approach makes it possible to verify industrially sized systems.

8.1 Introduction

The purpose of a manufacturing machine is to make products, which requires physical manufacturing processes to be carried out. In complex manufacturing machines, different types of products are concurrently being processed. Each type of product requires different manufacturing processes. To account for these recipe dependencies, a scheduling-based control approach is proposed in this thesis ([14]). The manufacturing steps to be performed are referred to as tasks, whereas the parallel operating mechatronic systems are referred to as resources. Supervisory Machine Control (SMC) is responsible for deciding when to do which tasks on which resources.

In complex manufacturing machines, many options exist to deploy the available resources to perform tasks that lead to the desired manufacturing purpose. Hence, such machines will exhibit a huge number of different machine behaviors that cannot all be tested beforehand. The choices in these machines can be categorized in three areas: which tasks to do, which resources to assign to them and which sequence of tasks to do at each resource (see Chapter 2, [12]). Conversely, the machine hardware imposes physical restrictions on the freedom for choices, e.g. with respect to material logistics (see Chapter 3, [13]). The freedom for choices that is applicable for some manufacturing request and

for some machine with its logistic restrictions can be defined in a scheduling model or system definition.

The system definition outlines the feasible behavior of the system, which still allows for invalid behavior. Non-FIFO behavior can be an example of invalid behavior: material leaves the machine in another order than it entered the machine. Another problem is deadlock. For instance a product must enter a particular machine unit that must first be emptied. But the machine unit can only be emptied along the route that the product is blocking. This is a small instance of, so-called, circular waiting situations of resources that have become a real problem in automated complex manufacturing systems.

Literature describes many approaches to avoid a manufacturing system to stop manufacturing due to deadlocks. For an extensive overview we refer to [4]. However, to the best of the authors' knowledge, the expressivity of their modelling approach is too limited for complex manufacturing machines. Some consider only a fixed routing per product [2, 17, 18, 19], or at least a predefined sequence of manufacturing processes [16]. In complex manufacturing machines, the manufacturing sequence can be flexible and even choices with respect to which manufacturing processes to carry out may be present. The scheduling-based modelling approach presented in Chapters 2 and 3 ([12, 13]) captures this scheduling flexibility. The deadlock-avoidance techniques used in [2, 17, 18, 19] cannot be proven to be correct in the more flexible domain of complex manufacturing machines. Also deadlock resolution as described in [20, 22, 23] is not desirable as deadlocks may not be locally resolvable due to the lack of buffer places. As a consequence, we have to develop another approach to verify deadlock absence in the scheduling problem defined in Chapters 2 and 3 ([12, 13]).

In Chapter 3 ([13]), validity constraints are described to avoid invalid behavior, especially deadlocks. First, constraints on the number of material instances residing at a set of resources are defined, like in [2]. These are called *maxWIP constraints*. Second, *tied precedences* are formulated, which express that tasks must be executed after each other without being interrupted, like in [19]. These constraints are formulated in this way, because they can easily and very quickly be checked. The scheduler must only verify that each action that it prescribes to the machine will never lead to a situation that invalidates a constraint.

The purpose of this chapter is to show how to verify that validity constraints indeed always avoid invalid machine behavior. This chapter concentrates on the deadlock absence property.

SMC uses guiding heuristics to quickly find a good schedule with respect to timing performance. Such heuristics compare the time effects of scheduling alternatives. In Fig. 8.1, the possible schedules are depicted as a search tree. The total tree covers all schedules representing physically feasible behavior (A). The validity constraints restrict this space to result in the space of valid behavior (B), which in turn is restricted by guiding heuristics to result in the space of good behavior (C). As mentioned before, the purpose of this chapter is to verify that SMC avoids invalid behavior, more specifically deadlocks. It may seem to be sufficient to explore area C for deadlocks. However, in practice it is not adequate to check area C. The guiding heuristics use predicted timing information that in the real-time environment of SMC can differ from the actual timing. This difference can trigger a rescheduling action, in which the guiding heuristics might choose another schedule due to the different timing. As a consequence, area C can drift. However, it will always stay within area B. Therefore, area B must be explored on the basis of the system definition and the validity constraints.

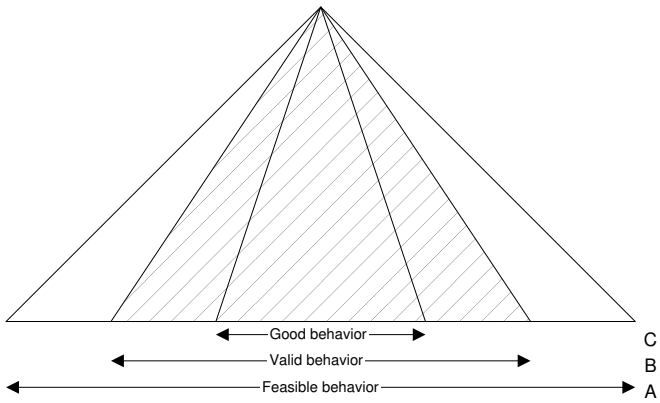


Figure 8.1: Search tree visualization of state space

It is technically convenient to use a directed graph, also called a *state space* or *transition system*, to represent the behavior of complex manufacturing machines. In essence, checking for the absence of deadlocks boils down to verifying whether each state has at least one outgoing edge until manufacturing is finished. Due to several effects, the state space can grow exponentially, which is called *state-space explosion*. This is partly due to parallelism in the system. If in some state n tasks can be executed in parallel, interleaving implies $n!$ possible execution sequences. Other exponential effects originate from the combinatoric structure of the scheduling problem. For instance m equivalent resources may be available in the machine, which is for instance the case in the Generalized Job Shop scheduling problem [21]. If a product visits such a resource n times, m^n different schedules are possible. Another combinatoric scheduling example is a set of n tasks having no precedence relation that must be executed by the same resource. These tasks can be scheduled in $n!$ different sequences, like in the Travelling Salesman scheduling problem [10].

These exponential effects often make it practically impossible to perform an exhaustive state space traversal for industrially sized problems. One approach to limit the state-space explosion is to manually tailor the abstraction level of the model to the property to be verified, and prove that the different models that are used for the different properties are behaviorally consistent, as in Chapter 7 ([8]).

In this chapter, we take a radically different approach. We have developed a general tool into which we can feed system definitions and validity constraints and which outputs whether the constraints are sufficient to guarantee absence of deadlock. We use our specific knowledge about the nature of manufacturing machines to make reductions of the state space in such a way that bad behavior is preserved and can be detected in the reduced state space.

The first reduction that we apply is to give some actions priority and ignoring the other actions in certain states. This can be justified using a confluence argument [7]. The effect of this operation is that for independent tasks only one particular order needs to be investigated in which they can occur. All other permutations of this ordering can be ignored, which is similar to partial order reduction [5].

The second reduction is based on the symmetry between equivalent resources that have the same material capacity and the same flow of material. For any machine model, it does not matter via which resource products are processed. Technically spoken, both options are strongly bisimilar [11, 15], and it is allowed to explore only one.

The third reduction is based on the irrelevance of the scheduled sequence of certain tasks that have no logistic effect for the deadlock absence property. In this case, a confluence argument can be used as well, to show that it is sufficient to explore only one schedule to investigate all possible deadlocks.

Depending on the particular circumstances, the reduction techniques can reduce the state space exponentially, preserving the deadlocks that may be present in the behavior of the machine. Using manufacturing facilities with different layouts we show the effect of the reductions. The tool is currently in use to analyze the designs of various wafer scanners and regularly finds unexpected deadlock situations. At the end of the chapter we show its applicability using two of such examples.

The structure of this chapter is as follows. In Section 8.2, a manufacturing system is defined using a static and a dynamic system definition. The behavior of such systems is defined as a transition system. Based on this transition system the validity constraints as well as the deadlock property are defined. Section 8.3 describes three specific cases in which state-space reduction techniques can be applied. The reduction effect of each of the three reduction techniques is illustrated using typical example manufacturing systems in Section 8.4. Moreover, several instances of a wafer scanner from the semiconductor industry [1] are used to show the applicability of the approach in industrial practice. Finally, concluding remarks are presented in Section 8.5.

8.2 Definition of the scheduling model as a transition system

We use the manufacturing model from Chapters 2 and 3 ([12, 13]). This model consists of a static part and a dynamic part. The static part defines the machine-specific restrictions imposed by the hardware of the machine, which is resource related. It describes which capabilities the resources can provide, how many material instances can reside on them, and which logistic transports are possible.

Definition 8.2.1 (Static system definition) *A static system definition is a 5-tuple* $\Sigma = (R, C, A, R_m, M_f)$*, where*

- R *is a given set of available resources;*
- C *is a given set of capabilities;*
- $A : R \to C$ *is a function that gives the capability which a resource can provide;*
- $R_m : R \to \mathbb{N}$ *is a function that gives the maximum number of material instances that can reside on a resource, called the material capacity;*
- $M_f : R \to \mathcal{P}(R)$ *is a function that gives the resources to which material can be transported from a certain resource.*

A simple example called *Simple Machine* is depicted in Fig. 8.2. The machine consists of two equivalent main processing resources: P0 and P1. Furthermore, products can be transported from the environment (E0) to a processing resource using a robot (R0). At the processing units, a cleaning resource (C0) is available to clean products before processing. Note that although the cleaning unit is essential in the production process, products will never be put on the cleaning resource itself. The robot and the processing resources can each contain one product (denoted between brackets in Fig. 8.2). The arrows in Fig. 8.2 depict how products can be transferred from resource to resource. The static system definition of *Simple Machine* looks as follows:

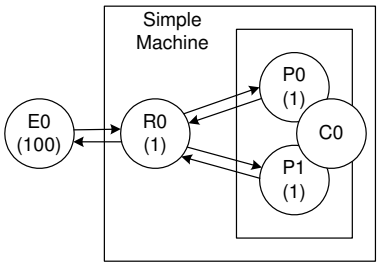


Figure 8.2: Layout of a simple manufacturing machine

- $R = \{\mathsf{E0}, \mathsf{R0}, \mathsf{P0}, \mathsf{P1}, \mathsf{C0}\}$;
- $C = \{\mathsf{E}, \mathsf{R}, \mathsf{P}, \mathsf{C}\}$;
- $A = \{(\mathsf{E0}, \mathsf{E}), (\mathsf{R0}, \mathsf{R}), (\mathsf{P0}, \mathsf{P}), (\mathsf{P1}, \mathsf{P}), (\mathsf{C0}, \mathsf{C})\}$;
- $R_m = \{(\mathsf{E0}, 100), (\mathsf{R0}, 1), (\mathsf{P0}, 1), (\mathsf{P1}, 1), (\mathsf{C0}, 0)\}$;
- $M_f = \{(\mathsf{E0}, \{\mathsf{R0}\}), (\mathsf{R0}, \{\mathsf{E0}, \mathsf{P0}, \mathsf{P1}\}), (\mathsf{P0}, \{\mathsf{R0}\}), (\mathsf{P1}, \{\mathsf{R0}\})\}$.

The dynamic system definition describes the tasks to do for a certain manufacturing request. They can recursively be structured in clusters and groups (see Chapter 2 [12]). The set of all tasks, clusters and groups are the nodes. A cluster indicates that if the cluster node is chosen all children of the cluster must be chosen, whereas a group has an attribute defining the allowed numbers of children to be chosen. Such an allowed number can be less than the number of children, which makes it possible to bypass tasks. Furthermore, as defined later, precedences of nodes cannot cross group boundaries, but they can cross cluster boundaries.

For each task the set of involved capabilities is defined, and at which capability which material instances reside at the beginning and at the end of the task. This, together with the precedence relation between nodes, outlines the room for choices with respect to task order.

Definition 8.2.2 (Dynamic system definition) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition. Then a dynamic system definition is a 12-tuple $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$, where*

- *T is a given set of tasks;*
- *G is a given set of groups;*
- *L is a given set of clusters;*

  Now we define N, the set of nodes, as: $N = T \cup G \cup L$
- *$L_n : L \to \mathcal{P}(N)$ gives the nodes contained in a certain cluster;*
- *$G_n : G \to \mathcal{P}(N)$ gives the nodes contained in a certain group;*
- *$G_a : G \to \mathcal{P}(\mathbb{N})$ gives the allowed numbers of node alternatives to be selected from a group;*
- *$I : T \to \mathcal{P}(C)$ is a function that gives the capabilities required to perform a certain task;*
- *$P : N \to \mathcal{P}(N)$ is a function that gives the predecessors of a certain node;*
- *M is the set of material instances;*

- $C_b, C_e : T \times C \to \mathcal{P}(M)$ *are functions that give the material instances residing on one of the resources with a certain capability that is involved in a certain task at the beginning and end of that task, respectively;*
- $\hat{m}_s : R \to \mathcal{P}(M)$ *is a function that gives the material instances initially residing on a certain resource.*

We only consider static and dynamic system definitions that satisfy the following properties:

1. The nodes in the system have a hierarchical structure. For all nodes $n \in N$ it must hold that $n \notin anc(n)$. Here, anchestor function $anc : N \to \mathcal{P}(N)$ gives the nodes in which a certain node is contained. The set $anc(n)$ is the smallest set satisfying the following conditions:
   - if $n \in G_n(n')$ or $n \in L_n(n')$, then $n' \in anc(n)$;
   - if $n'' \in anc(n')$ and $n' \in anc(n)$ then $n'' \in anc(n)$;
2. No group has only 0 as allowed number. I.e. $G_a(g) \neq \{0\}$ for all groups $g \in G$;
3. The capabilities in $C_b(t)$ and $C_e(t)$ exist also in $I(t)$. I.e. for all tasks $t \in T$ and capabilities $c \in C$ if $C_b(t,c) \cup C_e(t,c) \neq \emptyset$ then $c \in I(t)$;
4. P contains no cycles. For each node $n \in N$ it must hold that $n \notin allsucc(n)$. The function $allsucc : N \to \mathcal{P}(N)$ gives *all* successors of a node. It is the smallest set satisfying the following condition:
   - for all nodes $n' \in N$, $n'' \in \{n\} \cup anc(n)$, if $n'' \in P(n')$ then $n' \in allsucc(n)$, and $allsucc(n') \subseteq allsucc(n)$;
5. There is no precedence relation between node alternatives in a group. For all groups $g \in G$ and nodes $n \in N$ it holds that if $n \in G_n(g)$ then $P(n) = \emptyset$;
6. Precedence relations do not cross group boundaries. For all groups $g \in G$ and nodes $n, n' \in N$ it is the case that if $g \in anc(n)$ then $n' \in P(n) \Rightarrow g \in anc(n')$ and $n \in P(n') \Rightarrow g \in anc(n')$;
7. All tasks in a group concern the same material:

   $$\forall t, t' \in T, g \in G \cap anc(t) \cap anc(t').\ mat(t) = mat(t').$$

   Function *mat* gives the materials involved with a node:

   $$mat(n) = \{m{\in}M \mid \exists t{\in}T, c{\in}C.\ n{\in}\{t\} \cup anc(t)\ \wedge\ m{\in}C_b(t,c) \cup C_e(t,c)\};$$

8. The subsets of material instances involved in a task remain the same from the beginning to the end of a task. I.e. for all tasks $t \in T$:

   $$\bigcup_{c \in I(t)} C_b(t,c) = \bigcup_{c \in I(t)} C_e(t,c).$$

   This constraint implies that only closed systems are considered where no material enters or leaves the system;
9. Initially, the material capacity of all resources is not exceeded. For any resource $r \in R$ it holds that $\#\hat{m}_s(r) \leq R_m(r)$. Here $\#S$ gives the number of elements in the set S.

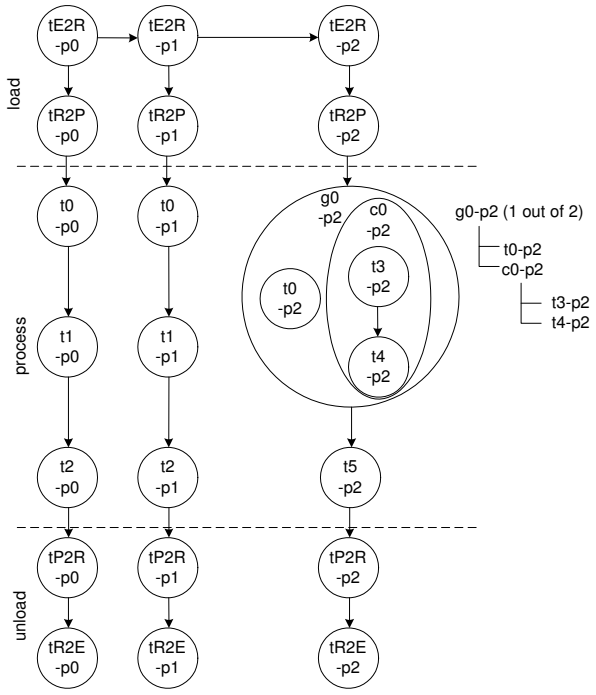


Figure 8.3: Three products to be produced by the simple manufacturing machine

As an example, we consider three products (p0, p1 and p2) to be manufactured by *Simple Machine*. The work to be done for these products is depicted in Fig. 8.3. By convention, node names end with a hyphen and the product id. Each product must be loaded, processed, and unloaded, respectively. Loading consists of an E2R task to transfer the product from the environment onto the robot, followed by an R2P task to transfer the product from the robot onto a processing unit, which is similar for unloading. The horizontal precedence arrows show that the products are loaded in the sequence p0, p1 and then p2. Products p0 and p1 are of the same type. They require a cleaning task (t0), and after that two processing tasks after another: t1 and t2, respectively. The cleaning of product p2, however, can also be done otherwise. Instead of the cleaning task, t0, cleaning can also be done by two successive other tasks, t3 and t4, that do not require the cleaning unit. Either one of the alternatives can be scheduled. In Fig. 8.3 this choice with respect to cleaning tasks is depicted as nested nodes in the graph, and additionally as a hierarchical node structure at the right. After cleaning, p2 requires one process task, t5. The dynamic system definition for this example looks as follows:

- T = {tE2R-p0, tE2R-p1, tE2R-p2, tR2P-p0, tR2P-p1, tR2P-p2, t0-p0, t0-p1, t0-p2, t3-p2, t4-p2, t1-p0, t1-p1, t2-p0, t2-p1, t5-p2, tP2R-p0, tP2R-p1, tP2R-p2, tR2E-p0, tR2E-p1, tR2E-p2};
- G = {g0-p2};
- L = {c0-p2};
- L_n = {(c0-p2, {t3-p2, t4-p2})};
- G_n = {(g0-p2, {t0-p2, c0-p2})};
- G_a = {(g0-p2, {1})};
- I = {(tE2R-p0, {E, R}), (tE2R-p1, {(E, R}), (tE2R-p2, {E, R}), (tR2P-p0, {R, P}), (tR2P-p1, {R, P}), (tR2P-p2, {R, P}), (t0-p0, {P, C}), (t0-p1, {P, C}), (t0-p2, {(P,

C}), (t3-p2, {P}), (t4-p2, {P}), (t1-p0, {P}), (t1-p1, {P}), (t2-p0, {P}), (t2-p1, {P}), (t5-p2, {P}), (tP2R-p0, {P, R}), (tP2R-p1, {P, R}), (tP2R-p2, {P, R}), (tR2E-p0, {P, E}), (tR2E-p1, {P, E}), (tR2E-p2, {P, E})};

- P = {(tR2P-p0, {tE2R-p0}), (t0-p0, {tR2P-p0}), (t1-p0, {t0-p0}), (t2-p0, {t1-p0}), (tP2R-p0, {t2-p0}), (tR2E-p0, {tP2R-p0}), (tE2R-p1, {tE2R-p0}), (tR2P-p1, {tE2R-p1}), (t0-p1, {tR2P-p1}), (t1-p1, {t0-p1}), (t2-p1, {t1-p1}), (tP2R-p1, {t2-p1}), (tR2E-p1, {tP2R-p1}), (tE2R-p2, {tE2R-p1}), (tR2P-p2, {tE2R-p2}), (g0-p2, {tR2P-p2}), (t4-p2, {t3-p2}), (t5-p2, {g0-p2}), (tP2R-p2, {t5-p2}), (tR2E-p2, {tP2R-p2})};
- P_t ={};
- $\mathcal{M}$ = {p0, p1, p2};
- C_b = {((tE2R-p0, E), {p0}), ((tE2R-p0, R), {}), ((tE2R-p1, E), {p1}), ((tE2R-p1, R), {}), ((tE2R-p2, E), {p2}), ((tE2R-p2, R), {}), ((tR2P-p0, R), {p0}), ((tR2P-p0, P), {}), ((tR2P-p1, R), {p1}), ((tR2P-p1, P), {}), ((tR2P-p2, R), {p2}), ((tR2P-p2, P), {}), ((t0-p0, P), {p0}), ((t0-p0, C), {}), ((t0-p1, P), {p1}), ((t0-p1, C), {}), ((t0-p2, P), {p2}), ((t0-p2, C), {}), ((t3-p2, P), {p2}), ((t4-p2, P), {p2}), ((t1-p0, P), {p0}), ((t1-p1, P), {p1}), ((t2-p0, P), {p0}), ((t2-p1, P), {p1}), ((t5-p2, P), {p2}), ((tP2R-p0, P), {p0}), ((tP2R-p0, R), {}), ((tP2R-p1, P), {p1}), ((tP2R-p1, R), {}), ((tP2R-p2, P), {p2}), ((tP2R-p2, R), {}), ((tR2E-p0, R), {p0}), ((tR2E-p0, E), {}), ((tR2E-p1, R), {p1}), ((tR2E-p1, E), {}), ((tR2E-p2, R), {p2}), ((tR2E-p2, E), {})};
- C_e = {((tE2R-p0, E), {}), ((tE2R-p0, R), {p0}), ((tE2R-p1, E), {}), ((tE2R-p1, R), {p1}), ((tE2R-p2, E), {}), ((tE2R-p2, R), {p2}), ((tR2P-p0, R), {}), ((tR2P-p0, P), {p0}), ((tR2P-p1, R), {}), ((tR2P-p1, P), {p1}), ((tR2P-p2, R), {}), ((tR2P-p2, P), {p2}), ((t0-p0, P), {p0}), ((t0-p1, P), {p1}), (t0-p0, C), {}), ((t0-p1, C), {}), ((t0-p2, P), {p2}), ((t0-p2, C), {}), ((t3-p2, P), {p2}), ((t4-p2, P), {p2}), ((t1-p0, P), {p0}), ((t1-p1, P), {p1}), ((t2-p0, P), {p0}), ((t2-p1, P), {p1}), ((t5-p2, P), {p2}), ((tP2R-p0, P), {}), ((tP2R-p0, R), {p0}), ((tP2R-p1, P), {}), ((tP2R-p1, R), {p1}), ((tP2R-p2, P), {}), ((tP2R-p2, R), {p2}), ((tR2E-p0, R), {}), ((tR2E-p0, E), {p0}), ((tR2E-p1, R), {}), ((tR2E-p1, E), {p1}), ((tR2E-p2, R), {}), ((tR2E-p2, E), {p2})};
- $\hat{m}_s$ = {(E0, {p0, p1, p2}), (R0, {}), (P0, {}), (P1, {}), (C0, {})}.

Given the two system definition parts, the physically feasible behavior of the machine carrying out the schedule is defined in the form of a transition system, which is often also called a state space or (behavioral) automaton, in Def. 8.2.7. The states represent which tasks have been executed and which material instances reside at which resources. The transitions indicate all allowed possibilities to go from one such state to another.

Before giving the main definitions two auxiliary concepts must be characterized. The predicate *successful*(n, tp) expresses whether node n is successfully executed given the successful tasks in tp. A task is successful if it occurs in tp. A cluster is successful if all its subnodes are successful. A group is successful iff an allowed number of nodes in the group are successful. If this number is zero, all predecessors of the group need to be successful.

Definition 8.2.3 *Let $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $n \in N$ be a node and $tp \subseteq T$ a set of executed tasks. We inductively define that n is successful in tp, notation successful(n, tp), iff*

- *if $n \in T$ is a task, then $n \in tp$;*
- *if $n \in L$ is a cluster, then for all $n' \in L_n(n)$ it must hold that successful(n', tp);*

- *if $n \in G$ is a group, then*

$$\#\{n' \in G_n(n) \mid \mathit{successful}(n', tp)\} \in G_a(n) \wedge$$
$$(\#\{n' \in G_n(n) \mid \mathit{successful}(n', tp)\} = 0 \Rightarrow \forall n'' \in P(n).\ \mathit{successful}(n'', tp)).$$

For a set of nodes np and a set of executed tasks tp the predicate $\mathit{successful}(np, tp)$ holds if for all nodes $n \in np$ $\mathit{successful}(n, tp)$ is valid.

The second auxiliary predicate that we define is *bypassed*. In order to define it, we must introduce two additional predicates. For a node $n \in N$ the expression $\mathit{successor}(n)$ gives the successor nodes of n that can immediately be executed after n. The definition is somewhat involved, because it can be that a direct successor of n is a group in which 0 tasks can be executed. Then the successor of such a group is also a successor of n.

Definition 8.2.4 *Let $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. The predicate $\mathit{successor}(n)$ for a node $n \in N$ is defined as the smallest predicate satisfying:*

$$\begin{aligned} \mathit{successor}(n) \;=\; & \{n' \in N \mid (n \cup \mathit{anc}(n)) \cap P(n') \neq \emptyset\} \cup \\ & \{\mathit{successor}(n') \mid n' \in \mathit{successor}(n) \cap G \ \wedge\ 0 \in G_a(n')\}. \end{aligned}$$

Another additional definition is the notion of *initiated* nodes. These are nodes that have already been started.

Definition 8.2.5 *Let $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. For a set of executed tasks tp the set of nodes $\mathit{initiated}(tp)$ is defined by*

$$\mathit{initiated}(tp) = \{n \in N \mid \exists t \in T.\ t \in tp \ \wedge\ n \in \mathit{anc}(t)\}.$$

The second auxiliary predicate that we define is $\mathit{bypassed}(t, tp)$ for a task $t \in T$ and a set of tasks $tp \subseteq T$. Its definition is quite involved. The predicate $\mathit{bypassed}(t, tp)$ holds for task t and set of tasks tp, if t is not successful (i.e. $t \notin tp$) and

- either a subsequent task is already successful,
- or the number of initiated nodes in a group containing that task is equal to the maximum number of nodes that can be successful in that group.

Definition 8.2.6 *Let $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $t \in T$ be a task and $tp \subseteq T$ a set of executed tasks. We define that t is bypassed, notation $\mathit{bypassed}(t, tp)$ iff*

$$\begin{aligned} t \notin tp \wedge \exists g \in G \cap \mathit{anc}(t).\ & \\ & \mathit{successor}(g) \cap tp \neq \emptyset \ \vee \\ & (((\mathit{anc}(t) \cup \{t\}) \cap G_n(g)) \setminus \mathit{initiated}(tp) \neq \emptyset \wedge \\ & \quad \#(G_n(g) \cap \mathit{initiated}(tp)) = \max(G_a(g))). \end{aligned}$$

Here $\max(S)$ for a finite set of natural numbers S is the largest number in S. The set $\mathit{bypassed}(tp)$ is defined as $\{t \in T \mid \mathit{bypassed}(t, tp)\}$.

This provides sufficient basic material to define the state space of a system.

Definition 8.2.7 (The state space of the system) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Then the state space is defined as the transition system $\Omega = (St, \hat{s}, Ac, \tau)$, where*

- *$St = (\mathcal{P}(T) \times (R \to \mathcal{P}(M)))$ are the states. The first component indicates the tasks that have been executed and the second component indicates which material instances can be found at which resource.*
- *$\hat{s} = (\emptyset, \hat{m}_s)$ is the initial state;*
- *$Ac = (T \times \mathcal{P}(R))$ are the labels of states indicating which task is executed at which resources;*
- *$\tau = \{((tp, ms), (t, r), (tp', ms')) \in St \times Ac \times St$*

$$
\begin{aligned}
&| \quad \mathit{successful}(P(t), tp) && (8.1)\\
&\wedge \quad t \in T \backslash (tp \cup \mathit{bypassed}(tp)) && (8.2)\\
&\wedge \quad \forall cap \in I(t).\ \exists res \in r.\ A(res) = cap && (8.3)\\
&\wedge \quad \forall res \in r.\ C_b(t, A(res)) \subseteq ms(res) && (8.4)\\
&\wedge \quad \forall res \in r, m \in M. && (8.5)\\
&\qquad m \in ms(res) \Rightarrow\ m \in ms'(res) \vee (\exists res' \in M_f(res).\ m \in ms'(res'))\\
&\wedge \quad \forall res \in R.\ \#ms'(res) \leq R_m(res) && (8.6)\\
&\wedge \quad tp' = tp \cup \{t\} && (8.7)\\
&\wedge \quad \forall res \in r.\ ms'(res) = (ms(res) \backslash C_b(t, A(res))) \cup C_e(t, A(res)) && (8.8)\\
&\wedge \quad \forall res \notin r.\ ms'(res) = ms(res)\} && (8.9)
\end{aligned}
$$

contains the transitions from state to state.

To make sure that the transitions conform to the intuition of the dynamic system definition, several cases must be distinguished. Cases one through three are properties that must hold for the task that is performed in the transition and cases four through seven are properties that must hold for the resources assigned to the task. Seven through nine express the update of the system state. Below each line of the definition of a state space is explained separately.

1. All predecessors of the task that is performed in the transition were successful before the transition.
2. The task to be performed did not happen and has not been bypassed.
3. For every capability involved in the task, there is a resource assigned to the task that can provide that capability.
4. The material to be used in the task is present at the resources that are assigned to the task. This check imposes *logistic flow integrity* as defined in Chapter 3 ([13]).
5. Material that is at an involved resource before a transition is at that same resource after the transition, or it has been transported to a resource to which it could be transported. This check imposes *logistic flow feasibility* as defined in Chapter 3 ([13]).
6. After a transition, the material capacity of all resources is not exceeded. This check imposes *material capacity feasibility* as defined in Chapter 3 ([13]).

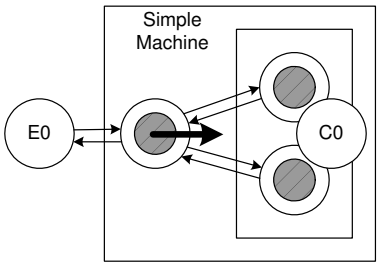


Figure 8.4: Deadlock in the simple manufacturing machine

7. The set tp' contains all tasks that have been performed before the transition and the task that has been performed during the transition.
8. After the transition, the material configuration of the resources involved in the task is the same as the material configuration at the beginning of the transition except for the changes made by the task.
9. After the transition, the material configuration of the resources not involved in the task is the same as the material configuration at the beginning of the transition.

As already argued in the introduction, it is possible that deadlocks occur in the state space. In the simple example, deadlock occurs when three products are loaded without one being unloaded first, as is depicted in Fig. 8.4. The product at resource R0 wants to move to either P0 or P1, whereas the products at resources P0 or P1 must move to R0.

In order to avoid such undesirable deadlocks, validity constraints are defined. A validity constraint is a predicate on states that expresses which states are and which are not accessible. The intention is that the validity constraints are defined such that deadlocks are avoided. Furthermore, these constraints are defined such that they can be quickly checked while dynamically scheduling a next task. We first define a constraint in an abstract sense.

Definition 8.2.8 *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition.*
Let $\Omega = (St, \hat{s}, Ac, \tau)$ be a state space. We call a function $VC : \tau \to \mathbb{B}$ a validity constraint. The constrained state space $\Omega_{VC} = (St, \hat{s}, Ac, \tau_{VC})$ has a constrained transition relation which is defined by

$$\tau_{VC} = \{(s, (t, r), s') \in \tau \mid VC((s, (t, r), s'))\}.$$

Note that the trivial validity constraint $VC(tr) = true$ for any $tr \in \tau$ keeps the original state space unchanged.

We explicitly define the notion of a deadlock state in a constrained state space as those states that do not have outgoing edges, but are also not finished. A finished state is a state where all tasks are either successful or bypassed.

Definition 8.2.9 (Finished and deadlocked states) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $\Omega = (St, \hat{s}, Ac, \tau)$ be a state space and $VC : \tau \to \mathbb{B}$ be a validity constraint. The set of finished states $F(\Omega_{VC})$ is defined as*

$$F(\Omega_{VC}) = \{(tp, ms) \in St \mid (\forall t \in T.\ \exists n \in \{t\} \cup anc(t).\ successful(n)\}.$$

The set of deadlock states $D(\Omega_{VC})$ is defined by:

$$D(\Omega_{VC})=\{(tp,ms)\in St \mid (tp,ms)\notin F(\Omega_{VC}) \wedge \forall t\in T, r\in\mathcal{P}(R), (tp',ms')\in St.\ ((tp,ms),(t,r),(tp',ms'))\notin\tau_{VC}\}.$$

Note that a state space of a production system never has a loop. This follows from the fact that in each transition the set *tp* is always extended with one task. A consequence of this is that a constrained state space without deadlocks always has finished states that will be reached. In other words, a constrained production machine works fine as long as deadlock freedom has been shown. In essence checking deadlock freedom is very easy by inspecting each state. The only problem is that the number of states can grow dramatically. In the next section techniques are provided to reduce the size of the state space maintaining the deadlocks in the system.

We define two concrete classes of validity constraints. The first one restricts the maximum number of material instances that are allowed to occupy a group of resources. These are called *maximum work in progress* constraints or *maxWIP* constraints. For the simple machine the maxWIP constraint that expresses that at most two material instances may occupy `R0`, `P0` and `P1` is sufficiently strong to prevent deadlocks. These constraints are formulated using the predicate $maxWIP_\Gamma$ where Γ is a set of pairs of the form (r,n). Here $r \subseteq R$ is a set of resources and $n \in \mathbb{N}$ is a natural number. It is formalized as follows:

Definition 8.2.10 (MaxWIP constraints) *Let $\Sigma=(R,C,A,R_m,M_f)$ be a static system definition, $\Delta=(T,G,L,L_n,G_n,G_a,I,P,M,C_b,C_e,\hat{m}_s)$ a dynamic system definition and $\Omega=(St,\hat{s},Ac,\tau)$ a state space. Let Γ be a set of pairs with a set of resources and a natural number. Then $maxWIP_\Gamma$ is the validity constraint defined as:*

$$maxWIP_\Gamma(((tp,ms),(t,r),(tp',ms'))) = \forall(r,n)\in\Gamma.\ (\sum_{res\in r}\#ms'(res)) \leq n$$

for every transition $((tp,ms),(t,r),(tp',ms')))\in\tau$.

Another way to avoid deadlocks is to use precedence constraints. The idea is that some tasks, or groups of tasks, can only be executed if it is guaranteed that another task can succeed it. The first task is called a tied predecessor of the second.

In the example of the simple production machine deadlock can be avoided by tying the movements of products onto the robot to the task of moving the product from the robot. In this way the robot always will become empty and available to move further products.

We use a function $P_t : N \to N \cup \{\perp\}$ that provides the tied predecessor of a node, unless there is no such a tied predecessor, in which case it yields $\perp$. We assume that P_t satisfies that if $P_t(n) = P_t(n')$ and $P_t(n) \neq \perp$, then $n = n'$.

Definition 8.2.11 (Tied precedence constraints) *Let $\Sigma = (R,C,A,R_m,M_f)$ be a static system definition and $\Delta = (T,G,L,L_n,G_n,G_a,I,P,M,C_b,C_e,\hat{m}_s)$ be a dynamic system definition. Let $\Omega = (St,\hat{s},Ac,\tau)$ be a state space. Let $P_t : N \to N$ be a tied predecessor function satisfying that $P_t(n)\in\bigcup_{m\in M}P_m(n)\cup\{\perp\}$ for all $n\in N$. Here, P_m is the part of P that concerns material m:*

$$P_m(n) = \{n'\in P(n) \mid m\in mat(n)\cap mat(n')\}.$$

Then the tied precedence constraint $tiedprecedence_{P_t}$ *is defined by:*

$$tiedprecedence_{P_t}(((tp, ms), (t, r), (tp', ms'))) = \\ \forall t' \in tp.\ (t \in ftsucc(t') \vee ftsucc(t') \backslash tp = \emptyset) \wedge carrythrough((tp', ms')).$$

We define $ftsucc(n)$ *as the first tasks in a tied successor node of* n*:*

$$ftsucc(n) = \{n'' \in firsttasks(n') \mid n' \in succ(n, P) \cap succ(n, P_t)\}.$$

Here, $succ(n, V)$ *gives the successors of a node* n*, given a precedence relation* V*:*

$$succ(n, V) = \{n' \in N \mid n \in V(n')\} \cup \\ \{n'' \in N \mid \neg \exists n' \in N.\ n \in V(n') \wedge \\ \exists n''' \in N.\ n \in G_n(n''') \cup L_n(n''') \wedge n'' \in succ(n''', V)\}.$$

If no nodes succeed n *directly, a node one level higher in the node hierarchy is looked at. Note that due to the hierarchical node structure, succ is well defined.*

The function firsttasks gives the first tasks in a node:

$$firsttasks(n) = \{t \in T \mid n \in \{t\} \cup anc(t) \wedge \neg \exists n' \in N.\ t \in allsucc(n') \wedge n \in anc(n')\}.$$

The predicate carrythrough indicates that all non-executed tied successor tasks of all executed tasks can be finished.

$$carrythrough((tp, ms)) = \\ \forall t \in tp, t' \in ftsucc(t) \backslash tp \Rightarrow \\ \exists r \in \mathcal{P}(R), tp' \in \mathcal{P}(T), ms' \in R \to \mathcal{P}(M). \\ ((tp, ms), (r, t'), (tp', ms')) \in \tau \wedge carrythrough((tp', ms')).$$

Note that as the state space has no cycles, carrythrough is well defined.

8.3 Checking deadlocks by reducing the state space

In order to know that maxWIP and tied precedence constraints guarantee that it is impossible to reach deadlock states, it is sufficient to generate all possible states of the machine that can be reached and to check that each such state is not a deadlock state (Def. 8.2.9).

This can very simply be achieved, except that the number of states in the machine is generally (larger than) astronomical. In this section, we present three ways to reduce the size of the state space, while maintaining potential deadlocks. For detailed proofs we refer to [3].

8.3.1 Interleaving of independent parallel tasks

The first reduction technique has to do with interleaving of independent parallel tasks. In some situations, no matter what interleaving is chosen during traversal, the same state will be encountered in the end. A general state-space reduction technique that can be applied in this case is the priorisation of actions [7], also called τ-priorisation. In literature, τ is used to denote internal transitions, which applies to all our transitions, because we are interested in deadlocks, and not in the particular nature of individual

transitions. It is allowed to apply priorisation of actions, if the system has no infinite behavior and is confluent. Our transition system has no infinite behavior, as with each step, an additional task is executed. Confluence is defined below. It says that in every state, where a transition from a confluent set of transitions can be chosen, and another transition is possible, a common state can be reached. The general definition of confluence is given in the form of a set of conditions on a confluent set of transitions (cf. [6]).

Definition 8.3.1 (Confluence) *Let $(St, \hat{s}, Ac, \tau)$ be a state space. Let $\mathbf{C} \subseteq \tau$ be a set of transitions. The set $\mathbf{C}$ is called τ-confluent if for all $(s, a, s') \in \mathbf{C}$ and $(s, a', s'') \in \tau$ it holds that:*

$$(\exists s''' \in St.\ (s', a', s''') \in \tau \wedge (s'', a, s''') \in \mathbf{C}) \ \vee \tag{8.10}$$
$$\exists a'' \in Ac.\ (s'', a'', s') \in \mathbf{C} \ \vee \tag{8.11}$$
$$\exists a''' \in Ac.\ (s', a''', s'') \in \tau \ \vee \tag{8.12}$$
$$s' = s'' \tag{8.13}$$

The prioritized state space is defined as follows. It says that Ω' can be constructed out of Ω by removing outgoing transitions from a state, as long as at least one transition from $\mathbf{C}$ remains. This generally reduces the size of a transition system substantially, especially, because many states become unreachable. We have the important theorem that says that if we apply τ-priorisation with a τ-confluent set, then the state space of the system has *exactly* the same deadlocks.

Definition 8.3.2 (τ-priorisation) *Let $\Omega = (St, \hat{s}, Ac, \tau)$ and $\Omega' = (St, \hat{s}, Ac, \tau')$ be state spaces. Let $\mathbf{C} \subseteq \tau$ be a set of τ-confluent transitions. We say that Ω' is a τ-prioritized reduction of Ω iff*

- $\tau' \subseteq \tau$;
- $\forall s, s' \in St, a \in Ac.\ (s, a, s') \in \tau \ \Rightarrow \ (s, a, s') \in \tau' \vee \exists s'' \in St, a' \in Ac.\ (s, a', s'') \in \tau \cap \mathbf{C}$.

A set of τ-confluent transitions $\mathbf{C}$ in the state space of a system as defined in the previous section is defined as follows. A more detailed explanation is given after the definition.

Definition 8.3.3 (Confluent transition set C) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $(St, \hat{s}, Ac, \tau)$ be the state space of the system. We define $\mathbf{C} \subseteq \tau$ to be the set consisting of the transitions $((tp, ms), (t, r), (tp', ms'))$ for which for all transitions $((tp, ms), (t', r'), (tp'', ms''))$ with $t \neq t'$ the following conditions hold.*

1. $\forall t'' \in T.\ mat(t'') \cap mat(t) \neq \emptyset \Rightarrow (t'' \cup anc(t'')) \cap allsucc(t) \neq \emptyset$;
2. $\forall res \in r.\ res \in unsafe \Rightarrow C_e(t, A(res)) \backslash C_b(t, A(res)) = \emptyset$;
3. $\forall g \in G, n \in (g \cup anc(g)).\ (anc(t) \cup \{t\}) \cap succ(n, P) = \emptyset$;
4. $G \cap anc(t) = \emptyset$.

The auxiliary functions used in this definition are defined directly below Def. 8.2.2 and in Def. 8.2.11, except for unsafe, which is given below. The set of unsafe resources contains those resources for which it is possible to put material onto it from more than one other location.

$$unsafe = \{res \in R \mid \sum_{res' \in R, res \in M_f(res')} R_m(res') > 1\}.$$

Condition one says that parallel tasks involving the same material are not necessarily confluent. Furthermore, the fact that in the end the same state will be encountered implies that no deadlock may be involved in the confluent set. If none of the transitions in the confluent set loads to an unsafe resource, i.e. a resource that can receive material from multiple other locations, confluence is not harmed. This is condition 2. Finally, the logistic effect of a group depends on which tasks are chosen in that group. Therefore, groups form an unreliable basis for confluence and are excluded as predecessor or relative of tasks in the confluent set. This yields conditions 3 and 4, respectively. Note that these conditions on set **C** are at the safe side. The set can be extended, but τ-confluence of an extended set has not been proven in [3].

8.3.2 Resource symmetry

The second reduction technique concerns equivalent resources in the system. In such a case multiple equivalent schedules can be generated, in which the only difference is that equivalent resources are swapped. The state space can be reduced in such cases using strong bisimulation [11], provided that it does not relate finished and deadlocked states. In such a case strong bisimulation reduction maintains deadlocks in our production machines, and hence, only the reduced system needs to be investigated.

Definition 8.3.4 (Strong bisimilarity) *Let $(St, \hat{s}, Ac, \tau)$ be a state space. For two states $s, t \in St$ to be strongly bisimilar, there must be a strong bisimulation relation R relating s and t, i.e. sRt. A relation R is a strong bisimulation relation iff it is symmetric, and for all states s, t such that sRt it holds that:*

$$(s, a, s') \in \tau \Rightarrow \exists t' \in St.\ (t, a, t') \in \tau \wedge s' R t'.$$

Two resources are symmetric in our transition system if they are of the same capability, have the same material capacity, and material can flow from and to the same resources. Note again that this definition can be extended, as e.g. also groups of resources can be symmetric, but this is not taken into account here.

Definition 8.3.5 (Symmetric resources) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition. We say that two resources $r \in R$ and $r' \in R$ are symmetric, notation $symm(r, r')$, iff*

$$\begin{gathered} A(r) = A(r') \wedge \\ R_m(r) = R_m(r') \wedge \\ M_f(r) = M_f(r') \wedge \\ \forall res \in R.\ r \in M_f(res) \Rightarrow r' \in M_f(res) \end{gathered}$$

Consider the state space $(St, \hat{s}, Ac, \tau)$ of the system. If there are two states $(tp, ms) \in St$ and $(tp, ms') \in St$ such that $\forall res \in R.\ ms(res) = ms'(res) \vee \exists res' \in R.\ symm(res, res') \wedge ms(res) = ms'(res')$ then it holds that (tp, ms) and (tp, ms') are strongly bisimilar.

Furthermore, it is easy to see that with this definition no deadlocked and finished states are related. So, this definition can then be employed in the following way. It is only necessary to investigate one state from the whole set of bisimilar states. So, in order to explore the full state space, it is only necessary to investigate only the first state encountered from the whole bisimilar set of states.

8.3.3 Non-logistic tasks

The third state-space reduction deals with mutually exclusive non-logistic tasks. A non-logistic task has no logistic effect as the materials stay at resources of the same capabilities. Tasks having no logistic effect can be executed in any arbitrary sequence without influencing deadlock behavior. Using a similar confluence argument as in Section 8.3.1 the state space can be reduced. We use the following confluent transition set.

Definition 8.3.6 (Confluent transition set D) *Let $\Sigma = (R, C, A, R_m, M_f)$ be a static system definition and $\Delta = (T, G, L, L_n, G_n, G_a, I, P, M, C_b, C_e, \hat{m}_s)$ be a dynamic system definition. Let $(St, \hat{s}, Ac, \tau)$ be the state space of the system. We define $\mathbf{D} \subseteq \tau$ to be the set consisting of the transitions $((tp, ms), (t, r), (tp', ms'))$ for which the following condition hold.*

$$anc(t) \cap NL \neq \emptyset \wedge \forall g \in G \cap anc(t).\ g \in NL.$$

Here, the definition of the non-logistic groups and clusters, notation

$$NL = \{n \in (G \cup L) \mid \forall t \in T, c \in C.\ n \in anc(t) \Rightarrow C_b(t, c) = C_e(t, c)\}.$$

This definition only considers groups or clusters containing only non-logistic tasks.

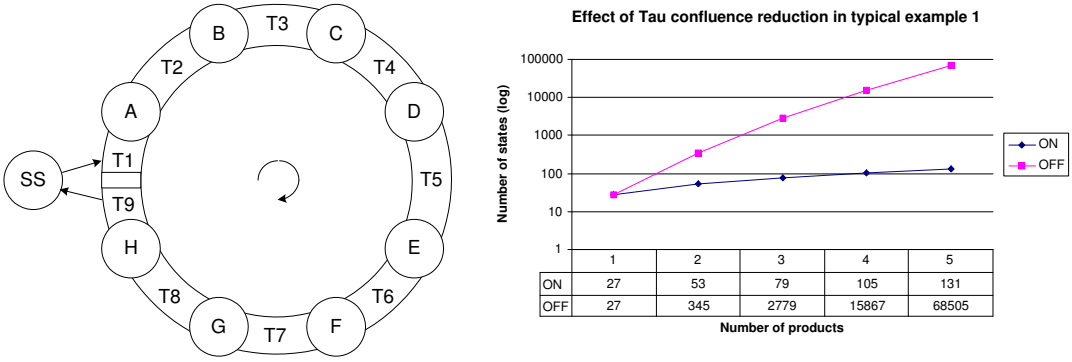
Using the fact that **D** is a confluent set of transitions, it is only necessary to investigate only one edge labelled with a non-logistic task in each state having such an edge. All other edges in this state can be ignored, while exactly finding all deadlocks in the system.

8.4 Results

This section presents the results that can be obtained by application of the reduction techniques presented in the previous section. A dedicated verification tool has been implemented in the C programming language. It traverses the state space as defined in Def. 8.2.7 and takes into account validity constraints as defined in Def. 8.2.10 and Def. 8.2.11. Furthermore, the state space reduction techniques as described in the previous section are implemented.

The explored system states $St = (\mathcal{P}(T) \times (R \to \mathcal{P}(M)))$ are stored such that memory usage is minimized. The states are put in a labelled structure. A bit array is used to indicate whether or not a task is passed, which implements $\mathcal{P}(T)$. Furthermore, an array of material locations is used to store the material configuration of the system, which implements $R \to \mathcal{P}(M)$. To be able to analyse an encountered deadlock state, it must be possible to generate a transition trace that led to the deadlock state. Therefore, the label of the state s that first led to a state s' is stored with state s', whereas the other transitions leading to s' are not stored at all.

This section first illustrates the reduction power of each of the three state-space reduction techniques using three typical example manufacturing systems. After that, applicability of the reduction techniques in industrial practice is illustrated using a wafer scanner [1].



	1	2	3	4	5
ON	27	53	79	105	131
OFF	27	345	2779	15867	68505

(a) Layout (b) Reduction power of interleaving reduction

Figure 8.5: Typical example 1: manufacturing cell

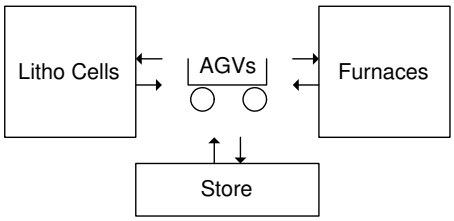
8.4.1 Reduction power

Interleaving of independent parallel tasks

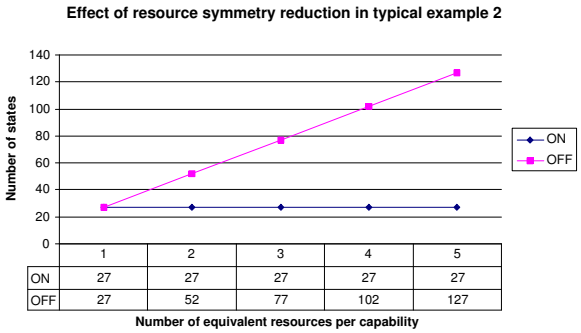
To illustrate the power of the reduction for interleaving of independent parallel tasks, the manufacturing cell depicted in Fig. 8.5(a) serves as an example. The cell consists of eight processing stations, A through H. Products enter the cell in a predefined sequence from unit SS, to which products also leave the cell. The SS unit and the processing stations are connected via a conveyor belt transportation system consisting of independent segments (T1 through T9). Each processing station and each conveyor belt segment can hold one product. Each product has to undergo 26 tasks. In case only one product is to be manufactured, 27 states are to be traversed, no matter which reduction technique is applied. In case multiple products are to be manufactured, many different interleaved traces are possible. However, interleaving of independent parallel tasks does not matter for deadlock. This is where the reduction for interleaving is very powerful, as is shown in Fig. 8.5(b). In the graph the number of states are shown with the τ-priorisation switched on and off. In this typical case, a state space with exponential size in the number of products is reduced to a linear state space.

Resource symmetry

The power of the resource symmetry reduction is illustrated using an example lithographic area in a semiconductor factory, as depicted in Fig. 8.6(a). The area is equipped with two types of processing equipment: litho cells and furnaces. Each batch of wafers has to undergo four processing cycles consisting of an exposure step in a litho cell and a bake step in a furnace. The wafer batches are transported using automatic guided vehicles (AGVs). Multiple instances of the processing and transportation equipment can be available. Including the transport from and to the store, each batch of wafers has to undergo 26 tasks, like in the previous example. In case only one instance of each type of equipment are available, 27 states are to be traversed for such a batch. However, in case multiple instances (resources) of each type of equipment (capability) is available, the number of traces (or schedules) that can be followed grows rapidly. As we do not store which resources are used in the passed tasks, the effect on the growth of the state space



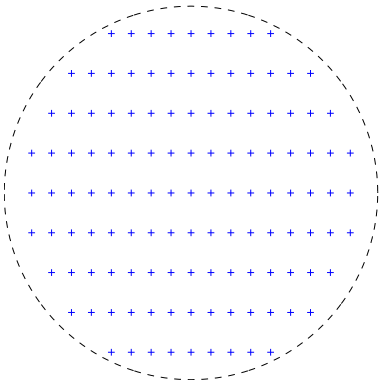
(a) Layout



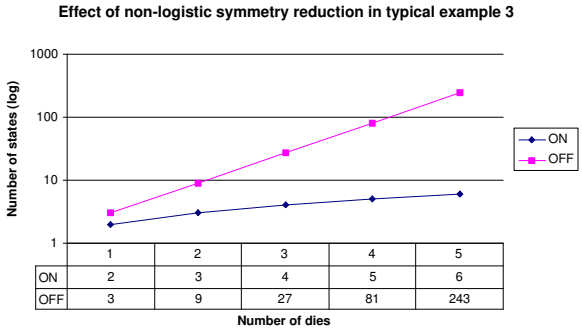
	1	2	3	4	5
ON	27	27	27	27	27
OFF	27	52	77	102	127

(b) Reduction power of resource symmetry reduction

Figure 8.6: Typical example 2: part of a semiconductor factory



(a) Layout



	1	2	3	4	5
ON	2	3	4	5	6
OFF	3	9	27	81	243

(b) Reduction power of non-logistic reduction

Figure 8.7: Typical example 3: integrated circuits on a wafer

is only linear in the number of equipment instances. However, for the deadlock property it does not matter which instance is chosen. Reduction for resource symmetry makes the state-space size independent of the number of equipment instances, as is shown in Fig. 8.6(b).

Non-logistic tasks

To illustrate the power of the reduction for non-logistic tasks, the exposure of integrated circuits (ICs) onto a wafer in a wafer scanner serves as an example. During exposure, the wafer is carried by a wafer stage. Each wafer can contain over a hundred ICs. In Fig. 8.7(a) the center coordinates of the ICs on a typical wafer are depicted by '+' signs. SMC is free to choose in which order and in which direction (up or down) to scan the ICs such that the fastest route results. The number of routes and states is exponential in the number of ICs. However, which route is taken is not of importance for the verification of the absence of deadlock, as the wafer stays at the same resource, which is called the wafer stage. This is where the reduction for non-logistic tasks is very powerful, as is shown in Fig. 8.7(b). An exponential size of the state space in the number of ICs is reduced to a

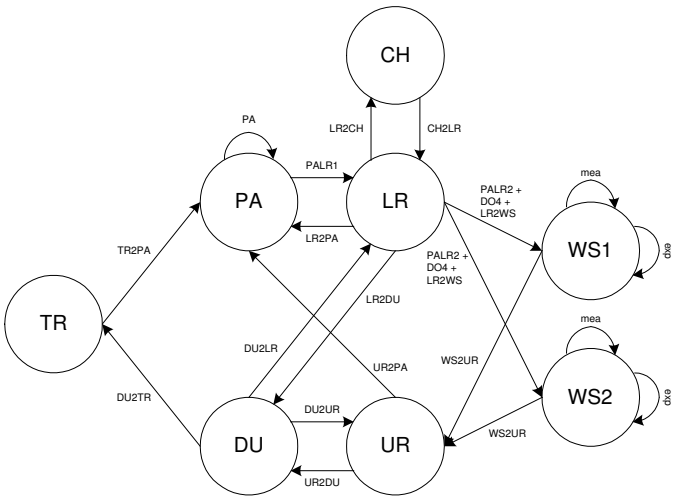


Figure 8.8: Layout of the wafer scanner with carrier handler: first industrial case

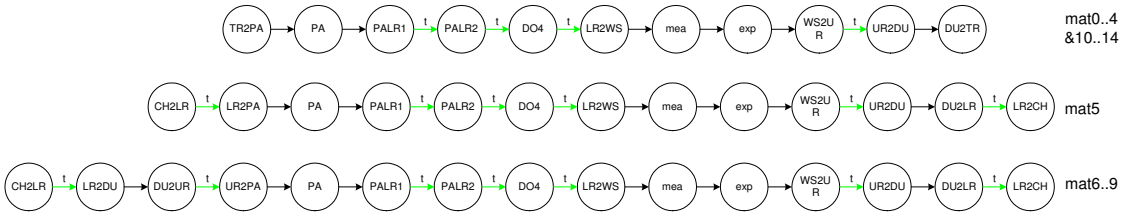


Figure 8.9: Different routes for first industrial case

linear one.

Practical use

The applicability of the reduction techniques is illustrated using two example wafer scanners from industry. The first example concerns an existing wafer scanner, the ASML TWINSCAN. The second example concerns a wafer scanner under development at ASML that uses Extreme Ultra Violet light for exposure, which is therefore called the EUV machine.

TWINSCAN

A layout of the TWINSCAN is depicted in Fig. 8.8. Typical for the TWINSCAN system is that it contains two wafer stages, shown at the right of Fig. 8.8: WS1 and WS2. Wafers can enter and leave the system via two different resources. By default, wafers enter the machine via the track (TR), and undergo the tasks depicted at the top of Fig. 8.9. In this figure, tied precedence edges are labelled with a 't'. Note that the routing details as described in the third example of the previous section are left out here. It is also possible to bring wafers into the machine via the carrier handler unit (CH). In this case, the wafers can follow two paths, as depicted in the middle and the bottom of Fig. 8.9. As an example, we consider the processing of three lots, each consisting of five wafers (mat0 through mat14). Which path which wafer must follow is depicted at the right of Fig. 8.9. When we verify this case, we find several deadlock states. For one of them, the material configuration in the deadlock state is depicted in Fig. 8.10, showing

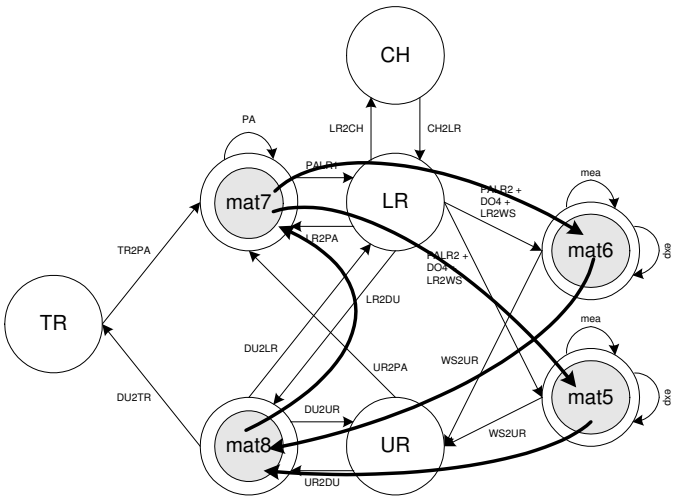


Figure 8.10: Deadlock state in first industrial case without maxWIP constraint

a circular wait condition. An example trace that leads to this deadlock state is shown in Fig. 8.11. This figure contains Gantt chart bars as well as material location lines, colored per material instance. Such deadlock states can be avoided with the introduction of an additional *maxWIP* validity constraint, stating that the pre-alignment unit (PA), the wafer stages (WS1 and WS2), and the discharge unit (DU) together contain no more than three wafers. For the deadlock state in Fig. 8.10, mat8 would not have entered the system, and the deadlock would be avoided. Verification confirms that indeed only valid schedules remain in this case. An example schedule is shown in Fig. 8.12.

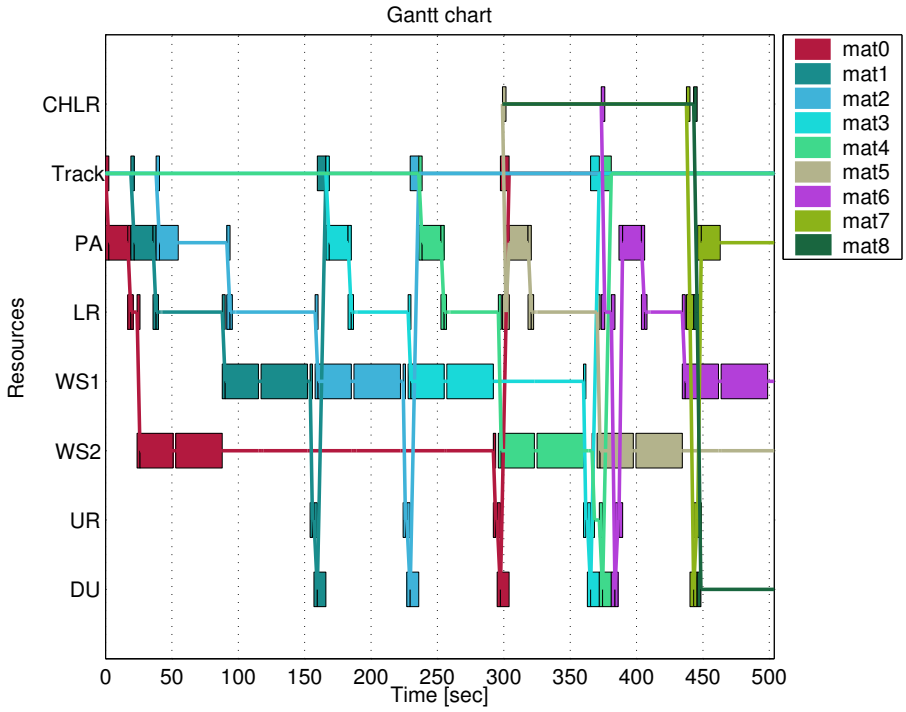


Figure 8.11: Deadlock trace for first industrial case without maxWIP constraint

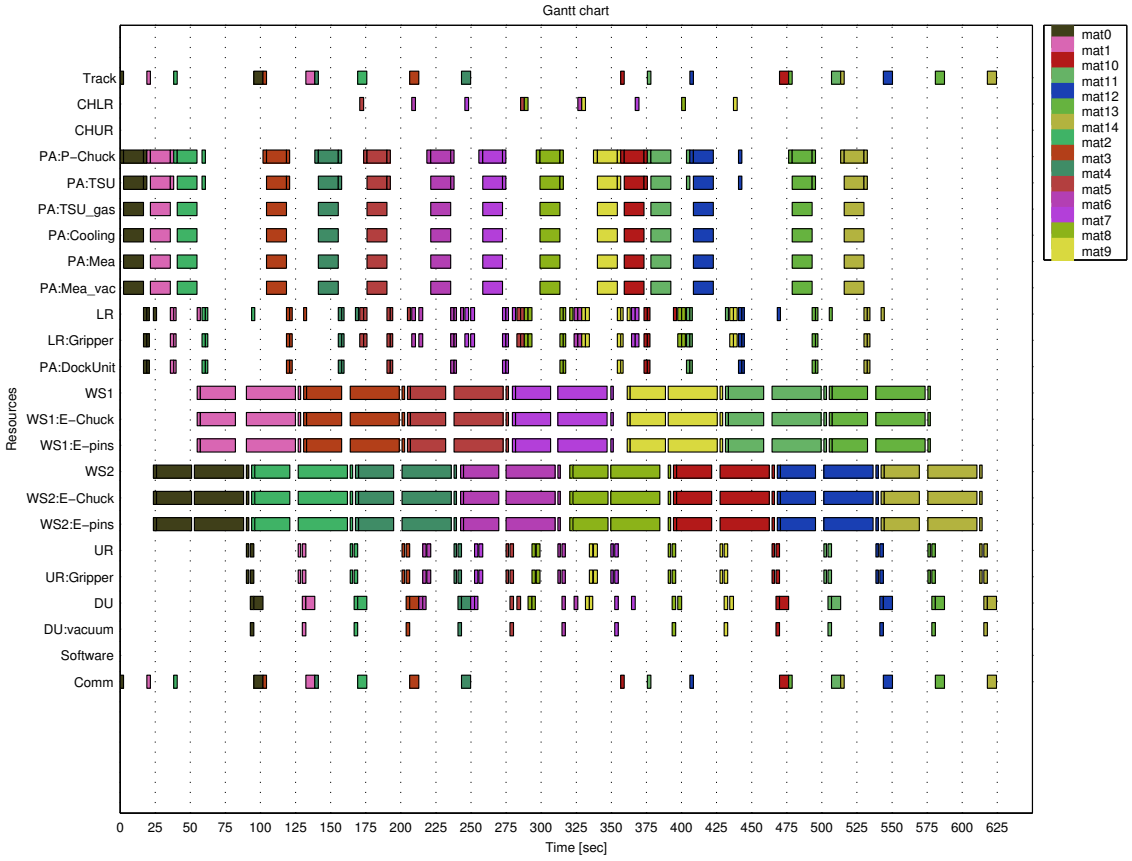


Figure 8.12: Schedule for first industrial case with additional maxWIP constraint

EUV wafer scanner

A layout of the EUV wafer scanner is shown in Fig. 8.13. As EUV light is absorbed by air, exposure must take place in vacuum. Therefore, the machine is equipped with two load locks (L1 and L2) to bring wafers entering the machine from atmospheric pressure to vacuum, and vice versa when leaving the machine. Furthermore, instead of two single-armed robots in TWINSCAN, only one combined robot with two arms (R1a and R1b) is available for wafer transportation. Again, two wafer stages are available (WS1 and WS2). Only loading from the track (TR) is considered in this example. Three *maxWIP* validity constraints are used, as depicted by the dotted squares and the numbers in Fig. 8.13. The absence of tied precedences and presence of multiple resources of the machine capabilities results in many scheduling possibilities. In case of five wafers, the state space consists of 67496 states. Application of the reduction techniques reduces the state-space size with one order of magnitude to 6746 states. Fig. 8.14 shows the influence of the number of wafers in the case on the state-space size, which still is exponential. Nevertheless, the reduction effect and memory usage efficiency of order of magnitude 10^7 states per gigabyte are sufficient to verify industrially sized problems.

8.5 Conclusions

A dedicated verification approach has successfully been developed to verify industrially sized cases of complex manufacturing machines for absence of deadlock. The cases are

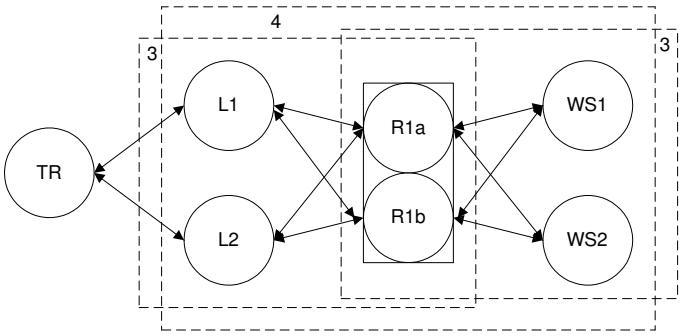


Figure 8.13: Layout of EUV machine: second industrial case

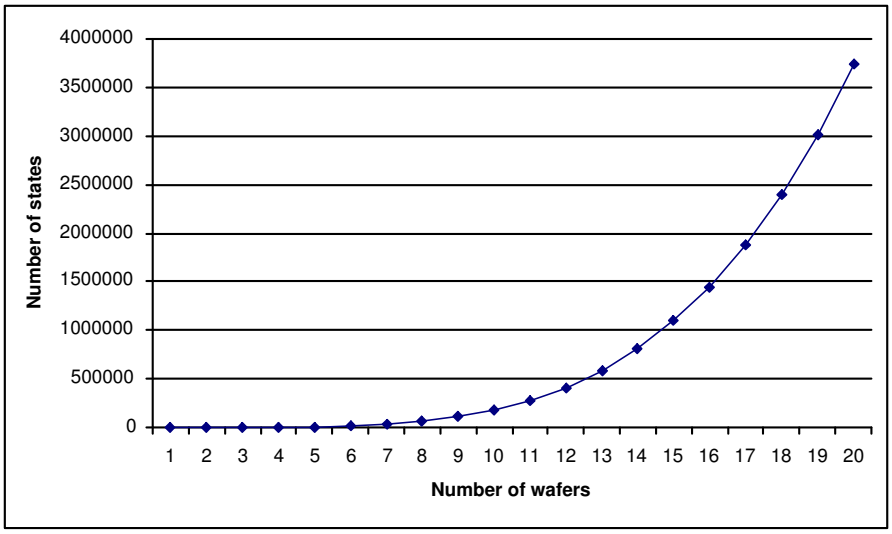


Figure 8.14: Number of states versus number of wafers in second industrial case

described by a definition of the manufacturing system (static system definition) and the work to be done (dynamic system definition), based on which a definition of the feasible state space is defined. Furthermore, a set of validity constraints that should avoid deadlock is specified. Several combinatoric effects cause the state space to grow exponentially, which makes it practically impossible to perform an exhaustive state space traversal for industrially sized problems.

As opposed to general paradigms to define system behavior such as automata or Petri nets, the system definition consists of lots of elements. The fact that elements have a specific and intuitive meaning makes it possible to formalize intuitive state-space reduction possibilities in terms of the system definition elements.

Three situations in which state space reduction techniques can be applied have been identified and formally defined. They can be justified by applying meta proofs to the specific characteristics of manufacturing systems and the deadlock absence property. A verification tool has been developed that can traverse the state space taking into account the validity constraints to detect deadlock states. The reduction techniques have been implemented, and memory usage is optimized.

Results show that two of the techniques can reduce exponential growth to linear growth, and one technique can make the state-space size independent of a linear effect. Furthermore, results of real wafer scanner problems show that the approach is suited for

verification of industrially sized problems.

The three reductions that we provide are rather straightforward. By making them more sophisticated, it should be possible to achieve further state-space reductions. Many other reductions are conceivable. An example is material symmetry. In case of identical product recipes, the state space can consist of many recurrent patterns that only differ in terms of material instances. These recurrent patterns can be mapped onto each other, as is e.g. done in [9].

The present work shows an effective approach to overcome state-space explosion problems by using specific knowledge of the nature of the type of systems and the properties to be verified to reduce state spaces. This approach has been used to effectively find and remove deadlock situations in a number of machines at ASML.

References

[1] ASML, 2004. Information on wafer scanners available through URL `http://www.asml.com/`, item: products - lithography.

[2] Z. A. Banaszak and B. H. Krogh. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Transactions on Robotics and Automation*, 6(6):724–734, December 1990.

[3] M. M. H. Driessen. Verification of task resource scheduling, June 2004. Internship report of Department of Computer Science, Eindhoven University of Technology, The Netherlands, available through URL `http://se.wtb.tue.nl/~bvdnieuw`.

[4] M. P. Fanti and M. Zhou. Deadlock control methods in automated manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 34(1):5–22, 2004.

[5] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In K. G. Larsen and A. Skou, editors, *Computer Aided Verification (CAV '91)*, number 575 in LNCS, pages 332–342. Springer, 1991.

[6] J. F. Groote and J. C. van de Pol. State space reduction using partial tau-confluence. In Nielsen, Mogens, and Rovan, editors, *Mathematical Foundations of Computer Science 2000*, number 1893 in LNCS, pages 383–393. Springer–Verlag, 2000.

[7] J. F. Groote and M. P. A. Sellink. Confluence for process verification. *Theoretical Computer Science B (Logic, semantics and theory of programming)*, 170(1-2):47–81, 1996.

[8] M. Hendriks, B. van den Nieuwelaar, and F. Vaandrager. Model checker aided design of a controller for a wafer scanner. In *Proceedings of the 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004)*, October 2004.

[9] M. Hendriks, N. J. M. van den Nieuwelaar, and F. W. Vaandrager. Recognizing finite repetitive scheduling patterns in manufacturing systems. In G. Kendall, E. Burke, and S. Petrovic, editors, *Proceedings of the Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA'03)*, pages 291–319. Automated Scheduling, Optimisation and Planning Group, University of Nottingham, UK, August 2003.

[10] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley-Interscience, Chichester, 1985.

[11] R. M. Milner. Calculus of communicating systems. *Lecture Notes in Computer Science*, 92, 1980.

[12] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, N. C. W. M. Braspenning, and J. E. Rooda. Predictive scheduling in complex manufacturing machines: scheduling alternatives and algorithm. *submitted to IEEE TAC*.

[13] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, N. C. W. M. Braspenning, and J. E. Rooda. Predictive scheduling in complex manufacturing machines: machine-specific constraints. *submitted to IEEE TSM*.

[14] N. J. M. van den Nieuwelaar, J. M. van de Mortel-Fronczak, and J. E. Rooda. Design of supervisory machine control. In K. Glover and J. Maciejowski, editors, *Proceedings of the European Control Conference 2003*, 2003. CD-ROM.

[15] D. M. R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings of the 5th GI-Conference*, number 104 in LNCS, pages 167–183. Springer–Verlag, 1981.

[16] J. Park and S. A. Reveliotis. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. *IEEE Transactions on Automatic Control*, 46(10):1572–1583, 2001.

[17] S. E. Ramaswamy and S. B. Joshi. Deadlock-free schedules for automated manufacturing workstations. *IEEE Transactions on Robotics and Automation*, 12(3):391–400, June 1996.

[18] S. A. Reveliotis, M. Lawley, and P. Ferreira. Polynomial-complexity deadlock avoidance policies for sequential resource allocation systems. *IEEE Transactions on Automatic Control*, 42(10):1344–1357, 1997.

[19] E. Roszkowska. Supervisory control for deadlock avoidance in compound processes. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 34(1):52–64, January 2004.

[20] N. Viswanadham, Y. Narahari, and T. L. Johnson. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using petri net models. *IEEE Transactions on Robotics and Automation*, 6(6):713–723, December 1990.

[21] M. Wennink. *Algorithmic Support for Automated Planning Boards*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1995.

[22] R. A. Wysk, N. S. Yang, and S. Joshi. Detection of deadlocks in flexible manufacturing cells. *IEEE Transactions on Robotics and Automation*, 7(6):853–859, December 1991.

[23] H. J. Yoon and D. Y. Lee. Deadlock-free scheduling of photolithography equipment in semiconductor fabrication. *IEEE Transactions on Semiconductor Manufacturing*, 17(1):42–54, February 2004.

CHAPTER

NINE

APPLICATIONS

Two of the applications discussed in this chapter have been protected in patent applications that were not mentioned earlier in this thesis.
The first application is a mechatronic idea to increase machine throughput for wafer scanners equipped with dual planar wafer stages, which is protected in patent application ASML ref. P-0346. The first filing was in Europe in September 2002, number 03255923.9. The subsequent filing in September 2003 was also done in other countries like the US and Japan.
The second application concerns synchronization of a wafer scanner and a track machine without a litho cell controller, using predictive information. This application is protected in patent application ASML ref. P-1818. The first filing was in the US in February 2004, number 10/781,945.

Applications

Whereas the previous chapters discussed the supervisory machine control concept, this chapter discusses application of the concept in industrial practice. Section 9.1 discusses diffusion of innovations in industry. In Section 9.2, the areas in which the results of this project can be applied are outlined. Section 9.3 discusses the issues involved in the roll-out of the developed supervisory machine control concept in embedded software of wafer scanners. A suitable roll-out strategy is sketched as well as the current status of the implementation at ASML. Section 9.4 concludes this chapter with a quantitative estimate of the benefits of the new concept compared to the current practice. It must be noted that this chapter contains no details for confidentiality reasons.

9.1 Diffusion of innovations

Whereas journal publications are common in the academic world, diffusion of innovations via text documents is not effective in industry. Therefore, lots of presentations have been given during this project. Theory has not been explained using mathematics but using metaphors. Timing and selecting have been explained using the brick game, and instantiating has been explained using the domino game. Besides disclosure of the theory, also a more structured way of thinking in the organization resulted from this. The benefits of application of the theory have been illustrated using small successes in relevant practical cases. Lots of pictures have been used and especially live demos using a prototype implementation, named *T-ReCS*: Task-Resource Control System. Most of the demos have been given in a simulation environment, but also control of a real reticle handler has been demonstrated. To be able to present the results of this demonstration that had to take place in a clean room, it was recorded on video tape. The video shows that the available resources can be deployed more efficiently than is done in the current control software by better exploitation of parallelism [1].

Whereas on the one hand lots of effort has been put in spreading information throughout ASML, on the other hand confidentiality of information had to be taken into account. Research partners had to sign non-disclosure agreement forms, and all documents leaving ASML had to be approved by the technical publication board. Most importantly, the intellectual property of all papers has been claimed by ASML in patent applications prior to publishing in the academic world. A patent application has been filed concerning scheduling as described in Chapters 2 and 3, and verification as described in Chapters 7 and 8 has been added afterwards. Also planning and reacting as described in Chapter 4 are protected in a patent application. A third patent application protects instantiating as described in Chapters 5 and 6. This project yielded two more patent applications, focussing on application of the theory. These patent applications are referred to in the next section.

[1] It must be noted that comparisons relate the new concept to the current practice. Some of the improvements can also be achieved without the new control concept.

9.2 Application areas

The theory described in this thesis can be applied for analysis as well as for real supervisory machine control. Some applications have been described in the examples of the previous chapters, but there are more. The ones of most practical importance are mentioned in the sequel.

Analysis

Analysis of machine behavior from a task resource point of view has proven to be very useful. Critical path analysis has, for instance, been applied to reduce software overhead and chuck swap time. It has also resulted in a mechatronic idea to increase machine throughput for machines equipped with dual planar wafer stages, that has been protected in a patent application [2]. Furthermore, the predicting part of the developed control concept can be used during design-time for simulation experiments. A prototype of a timing forecast tool covering the whole machine has been developed successfully. The behavior of the current real machine or of reference models can be matched accurately due to the flexible configuration possibilities. Results will especially be realistic if the real supervisory machine control would use the same predicting functionality.

Specifically for high-precision machines like wafer scanners, the developed prototype tool for instantiating kinematic calibration sequences [3] can be of great help during design. The design of calibration sequences is considered one of the most complex software development activities at ASML.

Real supervisory machine control

As described in the introduction, the specification of the supervisory machine control should form the basis for the real supervisory machine control software. In the project plan [1], automatic code generation from this specification was envisioned. This solves the problems of manual transformation from specification to implementation, which is labor-intensive and error-prone. However, another problem remains. The behavior of the machine is still tuned for a typical setting. Run-time interpretation and scheduling of the tasks specified in a TRS definition as described in this thesis and [6] also solves this problem.

The SMC concept developed can be applied to control a wafer scanner, as is shown in the cases in the previous chapters. According to this concept, separate software drivers of mechatronic systems (peripherals) can be controlled. Also, a hierarchical architecture consisting of master and sub SMC controllers can be thought of, as is described in [5]. Besides application in other types of machines, the scope under control of SMC can be broadened to a cluster of machines, e.g. a litho cell containing a wafer scanner and a track machine. An architecture consisting of only one controller can be thought of, or a hierarchical architecture with a litho cell master controller controlling a wafer scanner sub controller and a track sub controller. However, synchronization of both machines with their own controller without a master controller is also possible by exploiting the predictive information in the new concept. The latter idea is protected and described in another patent application [4].

9.3 Roll-out

Several issues complicate the implementation of the new SMC concept in ASML wafer scanners. This section describes these issues and a strategy how to cope with them. Furthermore, the current status of the implementation is described.

Strategy

First of all, there is the risk of malfunctioning caused by unforeseen problems that come with the introduction of a new control concept. A new control concept can only be introduced evolutionary to limit the risk involved with each step.

Secondly, making a step in the roll-out of the concept requires a certain investment, whereas the ASML software development resources are constantly under time pressure to deliver new functionality. Therefore, such required functional deliverables preferably serve as carriers to take steps in the introduction of the new control concept.

Thirdly, one should be very careful to change the behavior of the machine, even if the changed behavior does not violate the specification. The reason for this is that customers might rely on an unspecified part of the old behavior. Customers should, initially, not even notice that the machine is controlled using a new control concept. Therefore, introduction is first done without changes of interfaces and the new controller is configured such that it copies the old behavior.

Only after initial introduction of the new concept changes are planned to exploit more of the potential of the new concept. An example is the merging of control scopes to exploit scheduling possibilities. In the future also recovery search functionality can be envisioned. However, this is where the commercial issue comes in: any performance improvement must be paid for by the customer.

A road map covering the roll-out in several implementation steps in the coming years is under development. Besides the engineering activities for implementation in the embedded machine software, the road map also covers addressing open research issues as pointed out in the next chapter. Furthermore, a development traject is anticipated to connect research and engineering activities. Development activities focus on limiting risks in the main engineering stream and making the theory better accessible. To limit risks, the applicability of the concept in certain control scopes is validated in pilot activities to encounter unforeseen issues in an early stage. To make the theory better accessible, visual tools for configuration of the new SMC concept will be developed [7]. This makes application of the theory better scalable: adaptation of SMC for new functionality will often be restricted to changing its configuration, which could be done by non-experts then. Consequently, configuration management becomes more and more important.

Current status

At the time of writing this chapter, an initial implementation step has been taken. One of the order of magnitude 10 software modules responsible for production-related SMC has been transformed to the new SMC concept. Two functional deliverables that had to be implemented in this module have served as carriers. Furthermore, the limited complexity of its control scope and the fact that its behavior should be unchanged allowed for a light-weight implementation of the new concept: *T-ReCS Lite*.

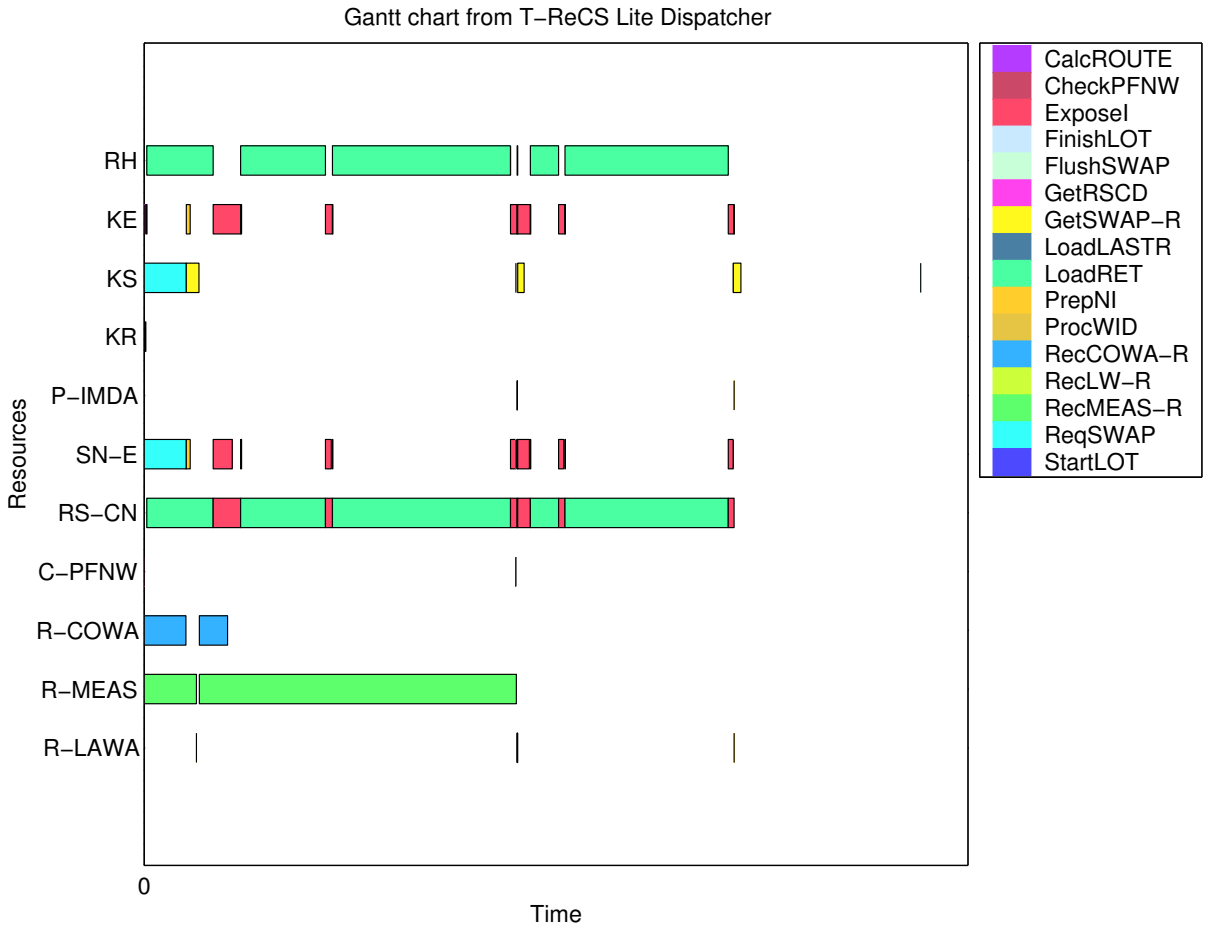


Figure 9.1: A schedule from T-ReCS Lite controlling a real machine

Nevertheless, the core elements of the new concept as described in Chapters 2 through 4 have already been applied. First of all, there is the higher abstraction level based on tasks and resources. Furthermore, a dispatching part and a predicting part are distinguished, the latter consisting of a planner and a scheduler. Also the exception recovery functionality using plan reconstruction has been applied, which appeared to be very powerful. Another elegant characteristic of the design is the clear division of control and data passing. Data is synchronized and passed through, but not via the control path.

In spite of the great timing pressure, the transformation has been performed very successfully. Only minor issues have been encountered during testing. Most of them did not concern control itself, but passing of data. The issues that did involve control concerned non-specified behavior that was implicitly present in the old implementation and that is being used for test purposes. Thanks to the configurability of the new concept these issues have been solved very quickly.

Figure 9.1 shows a Gantt chart obtained from a real machine. The timing depicted is real execution timing obtained from dispatcher tracing. The schedule concerns exposure of a lot consisting of two wafers each requiring three reticles.

9.4 Benefits

The benefits of the application of the new SMC concept can be classified in two main categories: development effort and machine performance. This section gives order-of-magnitude benefit estimates that are substantiated by extrapolation of preliminary facts and the expertise of several insiders at ASML.

Development effort

The configurability of the new concept has a very positive influence on the development effort required, both for implementation of changes in machine configuration as well as for changes in machine behavior.

An example case of a machine configuration change concerns a feasibility study of an alternative reticle handler configuration initiated by a cost of goods reduction programme. Reconfiguration of the *T-ReCS* prototype for this change took a few days. Changing the current implementation would take several months of development effort.

An example case of a change in machine behavior are the two deliverables that served as carriers for the introduction of *T-ReCS Lite* as mentioned in the previous section. Each of the two functional deliverables has been estimated to take a few months of development effort in the old implementation. The initial implementation of *T-ReCS Lite* also has taken a few months of development effort. However, after that, each of the functional deliverables has taken only a few days of effort.

Extrapolation of these preliminary results and taking some reservations into account, an SMC-related software effort reduction of factor 2 is expected to be a safe claim. In the type of market of ASML, the most important effect of this reduction lies in time to market. Taking the development budget into account, the ASML technology road map indicates that a development efficiency increase of factor 2 is indeed necessary to keep up with the required development pace.

Machine performance

Looking at machine performance, improvements lie in several areas. An important area is throughput, but besides that overall equipment efficiency (OEE) [8] plays a role.

Throughput

During this project, several example cases of throughput improvements were encountered, as for example presented at the 2002 ASML Technology Conference and in Chapters 2 and 3. These cases show throughput improvements of order of magnitude 10 %. However, these cases are quite specific, and their occurrence frequency is not known.

Nevertheless, for some other example cases of throughput improvement the overall effect on productivity can be estimated. One of those cases has been mentioned previously in this chapter in the context of the demonstration on the real reticle handler. The better exploitation of parallelism in that case results in an average productivity increase of order of magnitude 0.1 %. A throughput improvement of order of magnitude 0.1 % is equivalent with a machine market value increase of order of magnitude 10^4 euro.

In the area of reticle handling another case of throughput improvement can be identified. In the current implementation no more than two reticles are allowed on the turret

and the reticle stage to avoid deadlock. Application of the exact deadlock avoidance constraint as described in Chapter 7 allows for a faster load/unload schedule, overall leading to an average throughput improvement of order of magnitude 0.1 %.

Another case concerns the initial implementation of *T-ReCS Lite* mentioned in the previous section. Although not intended, this also resulted in an average productivity increase of order of magnitude 0.1 %. The old implementation is complex, and not fully transparent. As a consequence, potential parallelism was hidden, as opposed to the new concept in which parallelism is exploited automatically by design.

Overall equipment efficiency

Not all manufactured products meet the required quality, resulting in a quality efficiency loss. The new SMC concept can improve this in three ways. First of all, besides tuning for throughput, also tuning for product quality in critical circumstances is possible. This can be achieved by, e.g., scheduling extra calibrations, measurements, and other extra cautious processing behavior. Furthermore, the predictive functionality enables non-greedy behavior. This includes 'As Late As Possible' scheduling and time window variability reduction as is explained in Chapter 3, which has a positive effect on product quality. This is not possible in the old implementation. Finally, the improved exception recovery functionality can be applied to avoid rejection of wafers to result in a yield increase.

Sometimes the machine has to wait for wafers or reticles, resulting in a rate efficiency loss. The predicting functionality in SMC of the scanner can be used to synchronize better with the external logistics: the track and the reticle transportation system.

The availability of the machine is reduced by scheduled down time for preventive maintenance and unscheduled down time caused by interventions. The predictive functionality enables foreseeing gaps in the schedule that could be used for preventive maintenance, thus reducing scheduled down time. The improved exception recovery functionality can also be applied to avoid some of the interventions by automatic recovery. Furthermore, repair time can be reduced as better diagnostic information can be provided. In the current implementation, only the place *where* things went wrong can be traced (root error). Implementation of the pre- and post-conditions in terms of machine state as discussed in Chapters 5 and 6 could additionally provide information concerning *why* things went wrong (root cause). The avoidance of interventions and reduction of repair time result in a reduction of unscheduled down time. This is particularly important just after introduction of a new machine platform, when unscheduled down time is relatively high (time to quality).

Summing up and extrapolating these preliminary results, an overall productivity increase of order of magnitude 1 % is estimated to be well possible, without changes in the machine hardware. Such a productivity increase is equivalent with a market value increase of order of magnitude 10^5 euro per machine, with equal cost of goods.

References

[1] N. J. M. van den Nieuwelaar. Project plan: A framework for development of machine control systems, November 2000. Available through URL `http://se.wtb.tue.nl/~bvdnieuw`.

[2] N. J. M. van den Nieuwelaar. Skip zeroing of IF at exposure side, September 2002. Patent application ASML ref. P-0346, application number Europe: 03255923.9 (EP 1 404 712 A2), US: 10/665,351, China: 03125493.4, Japan: 2003-330017, Korea: 10-2003-0065530, Singapore: 200305621-5, Taiwan: 92125033.

[3] N. J. M. van den Nieuwelaar, W. H. G. A. Koenen, J. Onvlee, H. P. J. van Lierop, R. J. Dumont, M. A. R. Stoets, and J. E. Rooda. Run-time conditional sequencing mechanism for supervisory machine control, October 2003. Patent application ASML ref. P-1704, application number Europe: 03256456.9, Japan: 2004-286595.

[4] N. J. M. van den Nieuwelaar, J. Onvlee, and R. Boumen. Synchronization of wafer scanner and track to increase lithocell productivity, February 2004. Patent application ASML ref. P-1818, application number US: 10/781,945.

[5] N. J. M. van den Nieuwelaar, J. Onvlee, H. P. J. van Lierop, R. Boumen, R. J. Dumont, and J. E. Rooda. Supervisory machine control featuring dynamic scheduling, May 2004. Patent application ASML ref. P-1885, application number US: 10/852,678.

[6] N. J. M. van den Nieuwelaar, J. Onvlee, H. P. J. van Lierop, N. C. W. M. Braspenning, J.E. Rooda, M. M. H. Driessen, J. F. Groote, M. Hendriks, and F. Vaandrager. Run-time, model based supervisory control of manufacturing machines, December 2003. Patent application ASML ref. P-1784, application number US: 10/743,320.

[7] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Guide*. Addison Wesley, Reading, 1999.

[8] SEMI. Standard for definition and measurement of equipment productivity. Technical report, Semiconductor Equipment an Materials International, 2000. Available through URL `http://www.semi.org/`.

CHAPTER

TEN

Conclusions

This thesis focusses on problems being faced in the industrial practice of supervisory control of complex manufacturing machines (SMC). ASML wafer scanners serve as carriers. Problems lie in the area of development efficiency and machine performance. In the current wafer scanner, the control decision making software is interweaved with non-control functionality, and it is designed and implemented in a rigid way. Furthermore, formal specifications enabling verification often lack. However, the development cycles are shortening and machine configuration variability is increasing. This leads to more complex and error-prone software, more laborious implementation of new functionality, and makes software development more difficult to manage. The rigidness of the current SMC software also hinders tailoring the behavior of the machine for the specific demands of the product and customer, resulting in machine performance loss.

The main purpose of this PhD project was to develop a suitable formal method for specification of supervisory machine control. The specification should enable analysis and, moreover, should form the basis for the real SMC. This thesis describes how this purpose has been achieved using a scheduling-based specification approach. Furthermore, suggestions for further research are given.

10.1 Discussion

From the SMC point of view, a machine can be regarded as a task resource system (TRS). SMC decides when to do which tasks using which resources. In this thesis, we partition this decision-making process into three transformation phases: instantiating, selecting, and timing. The instantiating transformation defines a scheduling problem for a manufacturing request. The selecting transformation selects in the scheduling problem which tasks to do in which order using which resources. The timing transformation assigns start and finish times to the selected tasks, thus defining machine behavior.

The transformations are structured in a layered TRS framework shown in Fig. 1.4. The layers indicate that transformations rely on subsequent transformations in the sense that the effect of a decision made in some transformation on the machine behavior can only be evaluated by performing the consecutive transformations. Several TRS definition levels are distinguished, the level indicating the available room for choices. With each transformation certain decisions are made, and the TRS definition level decreases.

In this thesis, we specified the different TRS definition levels and the constraints that must be satisfied during the transformations in the TRS framework. This specification, together with the developed approaches to avoid state-space explosion enable analysis of SMC. Furthermore, we proposed a predictive-reactive control approach to embed the TRS framework in the dynamic environment of real SMC.

Specification and analysis

Concerning instantiating, we discussed two possible approaches: a non-deterministic and a deterministic approach. Both approaches result in an instantiated, unselected TRS definition, $\mathcal{D}_2$. The non-deterministic approach starts from the upper TRS definition level: the uninstantiated TRS definition $\mathcal{D}_3$. Tasks are instantiated from meta-tasks. Meta-tasks have pre-conditions and post-conditions on the system state. Pre-conditions define the instantiating constraints, whereas post-conditions define the expected state transformation after execution of an instantiated meta-task. We discussed general meta-tasks to process or transport products or other material, but also measurements and computations involved in the more specific domain of kinematic calibration. In the deterministic approach called planning, we defined the instantiating transformation in terms of planning rules. A sequence of predefined planning rules describes how manufacturing requests are detailed step by step to the task granularity matching with the resources under control ($\mathcal{D}_2$). We showed that both instantiating approaches can also be applied to recover from exceptions.

Concerning selecting, we proposed an intuitive way to outline which tasks can be done by which resources in which order. On the one hand, selection freedom is defined with respect to resource assignment and task order, but also with respect to which tasks are allowed to be selected in order to achieve the desired manufacturing purpose. On the other hand, selection restrictions are defined that are imposed by the physical layout of the machine. These restrictions ensure feasible behavior concerning material flow and buffering in the selected TRS definition ($\mathcal{D}_1$) resulting after this transformation. Within the feasible behavior, invalid behavior such as deadlock is still possible. We introduced additional validity constraints to avoid such invalid behavior.

Concerning timing, we showed that the hybrid nature of a manufacturing machine can be abstracted from for the purpose of SMC. Given a selected TRS that satisfies the constraints concerning physical resource interference ($\mathcal{D}_1$), we encapsulate the continuous behavior of the resources in between physical states in tasks. Given the begin and end physical state, we obtain the duration of each task by solving the differential equations describing the physical characteristics of the involved resources for that task. We show that, given the duration of tasks, the determination of the start and finish times of tasks ($\mathcal{D}_0$) can be regarded as a linear programming problem. Not only the task precedence relation, but also time windows can be described as linear constraints. Optimization can be applied not only to obtain a minimized total duration, but also to obtain non-eager behavior of certain tasks.

The TRS specification enables analysis of complex machines under SMC, e.g. to support design of sequences or to predict its behavior. To enable analysis by verification, we have applied two different approaches to overcome the state-space explosion problem. In the first approach, we use general model checking tooling and tailored models, whereas in the second approach we use a general model ($\mathcal{D}_2$) and a tailored model checker. In the first approach we verify for a subset of $\mathcal{D}_2$ time optimality of steady-state operation and deadlock absence. We use two consistent models of different abstractions that each are tailored for a specific property. In the second approach we verify deadlock absence in any $\mathcal{D}_2$ using characteristics of the property and model to apply state space reduction techniques. These approaches are in accordance with the message of the No Free Lunch Theorem: in areas suffering from combinatoric effects, dedicated approaches perform better than generic approaches.

Real supervisory machine control

We proposed a predictive-reactive scheduling approach to embed the TRS framework in the dynamic environment of SMC. In this approach, SMC consists of a predicting part and a dispatching part. The dispatching part is connected to the resources and dispatches tasks in a schedule obtained from the predicting part to the resources. The TRS framework is embedded in the predicting part in the form of TRS translation functions. We describe several scenarios to react to different types of control triggers using these TRS translation functions.

The presented planning function is suited to quickly react to triggers from the user or to exception triggers for which recovery is predefined. In case recovery is not predefined, it can be searched for using the instantiating functionality within the constraints of the uninstantiated TRS ($\mathcal{D}_3$). We proposed a scheduling algorithm combining the selecting and timing transformation that is suited for application in SMC. The algorithm applies heuristics to quickly find a good schedule, and enables partial schedule dispatching to avoid control overhead.

In the proposed concept of supervisory machine control by predictive-reactive scheduling the TRS definitions serve for configuration of SMC rather than as a source for code generation. This way, run-time predictive scheduling is possible to enable optimization of machine behavior for specific settings.

The contribution of this thesis lies in the effective application of theory from different areas rather than in the development of theory itself. We gathered ingredients from mechanical engineering, game theory, scheduling, computer science, and mathematics and integrated them into one framework. The framework makes the phenomena playing a role in SMC explicit in the form of TRS definition elements. Furthermore, the framework structures and relates the phenomena and their consequences on control decisions in the form of transformation constraints. Compared to current practice in machine control software, the complexity that was implicit and interweaved is now unravelled and localized.

The applications chapter shows that the framework is well applicable in industrial practice and that development effort and machine performance are indeed improved considerably. A safe estimate of the expected SMC-related software effort reduction is a factor 2, which is necessary to keep up with the development pace. A safe estimate of the expected machine performance increase is order of magnitude 1 %, which increases the gross margin per machine by order of magnitude 10^5 euro. The roll out of the new SMC concept at ASML is currently in progress.

10.2 Further research

As discussed before, this thesis combines theory from many expertise areas into one framework. Breadth as well as depth of exploration per expertise area has been chosen in a pragmatic way, such that a consistent mix and practically applicable package of theory resulted. In our opinion, this thesis forms a solid basis for further development and engineering to successively cash the involved benefits. Further exploration in lots of areas can be thought of. In this section we mention some directions for further research that we envision to be useful complements to the current work.

The first interesting research direction is optimization. In this thesis, the timing and selecting transformation layers have been integrated into a rather straightforward scheduling algorithm. Instantiating has only been applied in an isolated setting, such that only after instantiation of an entire TRS definition of level 2 the successive transformations were applied. Moreover, search algorithms were investigated to a limited extent only. Integration of all three transformation layers would enable more effective decision making with respect to instantiating. Moreover, it would be interesting to investigate application of more sophisticated optimization algorithms to all three integrated layers.

The second interesting area for further research is software architecture. In this thesis, a machine control concept has been proposed that can be applied as a control engine in embedded machine software. However, some software architectural questions remain. One of these questions is whether and how to split the total machine into different control scopes, and how to connect the different controllers into a composed architecture. Some options were illustrated during this project, but a theoretically founded approach would be useful. Another question is how to ensure a correct flow of data through the machine without bothering the control engine with passing through of data that it doesn't use itself. For the implementation of *T-ReCS Lite* a specific solution has been chosen, but a generic and theoretically founded approach would again be useful. The third question concerns reaction dynamics. This thesis describes several reaction scenarios, but the question remains which one to apply in which circumstances. Moreover, other reaction scenarios can be thought of. Furthermore, reaction scenarios in composed control architectures remain to be investigated.

The third interesting research direction is verification, which is understandable as this thesis only describes the mechanical engineering part of a larger project. The original project was intended for two PhD students: a mechanical engineer and a computer scientist. The second part of the project, the computer science part, has in principle not been undertaken. This thesis only pays attention to the case-specific verification of deadlock absence in the selecting transformation. It would be interesting to broaden this scope in several directions. In the context of the TRS framework, we think of verification for classes of cases, other properties and other transformations. We think, e.g., of proving deadlock absence for all possible manufacturing requests, timed properties, and reachability in instantiating, respectively. In the context of machine control we think of verification of reaction dynamics and behavior of composed control architectures.

BIBLIOGRAPHY

R. J. Abumaizar and J. A. Svestka, "Rescheduling job shops under random disruptions," *International journal of production research*, vol. 35, no. 7, pp. 2065–2082, 1997.

R. Alur, C. Courcoubetis, and D. L. Dill, "Model checking in dense real time," *Information and Computation*, vol. 104, pp. 2–34, 1993.

R. Alur and D. L. Dill, "Automata for modeling real-time systems," in *17th International Colloquium on Automata, Languages and Programming*, 1990, pp. 322–335.

R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

ASML, 2004, information on wafer scanners available through URL `http://www.asml.com/`, item: products - lithography.

R. J. Aumann and S. Hart, *Handbook of game theory: with economic applications*, Amsterdam, North-Holland, 2002.

J. C. M. Baeten and W. P. Weijland, *Process Algebra*, no. 18, Cambridge University Press: Cambridge Tracts in Theoretical Computer Science, 1990.

Z. A. Banaszak and B. H. Krogh, "Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 6, pp. 724–734, 1990.

D. A. v. Beek and J. E. Rooda, "Languages and applications in hybrid modelling and simulation: positioning of Chi," *Control Engineering Practice*, vol. 8, no. 1, pp. 81–91, 2000.

B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–341, 1994.

M. Browne, E. Clarke, and O. Grümberg, "Characterizing finite Kripke structures in propositional temporal logic," *Theoretical Computer Science*, vol. 59, no. 1,2, pp. 115–131, 1988.

R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transaction on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, Wiley, 1996.

H. Chen and B. Hu, "Schedule-driven supervisory control of flexible manufacturing systems," in *30th Conference on Decision and Control*, 1991, pp. 2186–2191.

Y. Chiu and M. Perng, "Self-calibration of a general hexapod manipulator using cylinder constraints," *International Journal of Machine Tools & Manufacture*, vol. 43, pp. 1051–1066, 2003.

H. Cho and R. A. Wysk, "Graph-theoretic deadlock detection and resolution for flexible manufacturing systems," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 413–421, 1995.

S. Chung, S. Lafortune, and F. Lin, "Limited lookahead policies in supervisory control of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1921–1935, 1992.

E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, The MIT Press, 2000.

D. Daney, Y. Papegay, and A. Neumaier, "Interval methods for certification of the kinematic calibration of parallel robots," in *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, 2004.

E. W. Dijkstra, "Cooperating sequential processes," in *Programming Languages: NATO Advanced Study Institute*, F. Genuys, Ed., pp. 43–112, Academic Press, 1968.

J. Dorn, R. Kerr, and G. Thalhammer, "Reactive scheduling," *International journal of human-computer studies*, vol. 42, pp. 687–704, 1995.

M. M. H. Driessen, "Verification of task resource scheduling," , 2004, internship report of Department of Computer Science, Eindhoven University of Technology, The Netherlands, available through URL `http://se.wtb.tue.nl/~bvdnieuw`.

M. P. Fanti and M. Zhou, "Deadlock control methods in automated manufacturing systems," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 34, no. 1, pp. 5–22, 2004.

A. Fehnker, "Scheduling a steel plant with timed automata," in *Proceedings of the sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*, IEEE Computer Society Press, 1999.

I. C. M. Flinsenberg, "Route planning algorithms for car navigation," Ph.D. dissertation, Eindhoven University of Technology, The Netherlands, 2004.

A. Garnaev, *Search games and other applications of game theory*, Springer, 2000.

S. Gaubert and J. Mairesse, "Task resource models and (max,+) automata," in *Idempotency*, J. Gunawardena, Ed., pp. 131–144, Cambridge, UK: Cambridge University Press, 1998.

S. Gaubert and J. Mairesse, "Modeling and analysis of timed Petri nets using heaps of pieces," *IEEE Transactions on Automatic Control*, vol. 44, no. 4, pp. 683–697, 1999.

B. Gebremichael and F. W. Vaandrager, "Control synthesis for a smart card personalization system using symbolic model checking," in *Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, K. G. Larsen and P. Niebert, Eds., no. 2791 in LNCS, Springer–Verlag, 2004, pp. 189–203.

R. J. v. Glabbeek and W. P. Weijland, "Branching time and abstraction in bisimulation semantics," *Journal of the ACM*, vol. 43, no. 3, pp. 555–600, 1996.

P. Godefroid and P. Wolper, "Using partial orders for the efficient verification of deadlock freedom and safety properties," in *Computer Aided Verification (CAV '91)*, K. G. Larsen and A. Skou, Eds., no. 575 in LNCS, Springer, 1991, pp. 332–342.

P. Gohari and W. M. Wonham, "Reduced supervisors for timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 48, no. 7, pp. 1187–1198, 2003.

G. H. Golub and C. F. V. Loan, *Matrix Computations*, Baltimore, ML: The John Hopkins University Press, 1983.

J. F. Groote and J. C. v. d. Pol, "State space reduction using partial tau-confluence," in *Mathematical Foundations of Computer Science 2000*, Nielsen, Mogens, and Rovan, Eds., no. 1893 in LNCS, Springer–Verlag, 2000, pp. 383–393.

J. F. Groote and M. P. A. Sellink, "Confluence for process verification," *Theoretical Computer Science B (Logic, semantics and theory of programming)*, vol. 170, no. 1-2, pp. 47–81, 1996.

H. Gueguen and M. Lefebvre, "A comparison of mixed specification formalisms," in *Automation of mixed processes: Hybrid Dynamic Systems: ADPM 2000*, Aachen: Shaker Verlag, 2000, pp. 133–138.

V. Hartonas-Garmhausen, E. M. Clarke, and S. Campos, "Deadlock prevention in flexible manufacturing systems using symbolic model checking," in *IEEE Conference on Robotics and Automation*, vol. 1, 1996, pp. 527–532.

M. Hendriks, N. J. M. v. d. Nieuwelaar, and F. W. Vaandrager, "Model checker aided design of a controller for a wafer scanner," Report NIII-R0430, Nijmegen Institute for Computing and Information Sciences, University of Nijmegen, The Netherlands, 2004a.
*http://www.cs.kun.nl/ita/publications/papers/fvaan/HNV04.html

M. Hendriks, B. van den Nieuwelaar, and F. Vaandrager, "Model checker aided design of a controller for a wafer scanner," in *Proceedings of the 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004)*, 2004b.

M. Hendriks, N. J. M. van den Nieuwelaar, and F. W. Vaandrager, "Recognizing finite repetitive scheduling patterns in manufacturing systems," in *Proceedings of the Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA'03)*, G. Kendall, E. Burke, and S. Petrovic, Eds., Automated Scheduling, Optimisation and Planning Group, University of Nottingham, UK, 2003, pp. 291–319.

J. M. Hollerbach, *A survey of kinematic calibration*, MIT Press, 1989.

J. M. Hollerbach and C. W. Wampler, "The calibration index and taxonomy for robot kinematic calibration methods," *International Journal of Robotics Research*, vol. 15, no. 6, pp. 573–591, 1996.

D. Jevtic, "Method and apparatus for automatically generating schedules for wafer processing within a multichamber semiconductor wafer processing tool," , 1997, patent no. US 6,201,999.

J. Kim, T. Lee, H. Lee, and D. Park, "Scheduling analysis of time-constrained dual-armed cluster tools," *IEEE Transactions on Semiconductor Manufacturing*, vol. 16, no. 3, pp. 521–534, 2002.

C. M. H. Kuijpers, C. A. J. Hurkens, and J. B. M. Melissen, "Fast movement strategies for a step-and-scan wafer stepper," *Statistica Neerlandica*, vol. 51, no. 1, pp. 55–71, 1997.

S. Kumar, N. Ramanan, and C. Sriskandarajah, "Robotic system control," , 2003, patent no. US 6,556,893.

K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1/2, pp. 134–152, 1997.

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Chichester: Wiley-Interscience, 1985.

M. Lawley and S. A. Reveliotis, "Deadlock avoidance for sequential resource allocation systems: Hard and easy cases," *International Journal of Flexible Manufacturing Systems*, vol. 13, no. 4, pp. 385–404, 2001.

M. Lawley, S. A. Reveliotis, and P. Ferreira, "Design guidelines for deadlock handling strategies in flexible manufacturing systems," *International Journal of Flexible Manufacturing Systems*, vol. 9, no. 1, pp. 5–30, 1997.

Y. Li and Z. H. Lin, "Supervisory control of probabilistic discrete-event systems with recovery," *IEEE Transactions on Automatic Control*, vol. 44, no. 10, pp. 1971–1975, 1999.

J. M. Maciejowski, *Predictive control with constraints*, Harlow: Prentice Hall, 2002.

S. Manetti and M. C. Piccirilli, "A singular-value decomposition approach for ambiguity group determination in analog circuits," *IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications*, vol. 50, no. 4, pp. 477–487, 2003.

H. Marchand, O. Boivineau, and S. Lafortune, "On the synthesis of optimal schedulers in discrete-event control problems with multiple goals," *SIAM Journal on Control Optimization*, vol. 39, no. 2, pp. 512–532, 2000.

K. L. McMillan, "Symbolic model checking," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, 1992.

R. M. Milner, "Calculus of communicating systems," *Lecture Notes in Computer Science*, vol. 92.

K. Miyashita and K. Sycara, "CABINS: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair," *Artificial Intelligence Journal*, vol. 76, no. 1-2, pp. 377–426, 1995.

D. C. Montgomery and G. C. Runger, *Applied Statistics and Probability for Engineers*, New York: John Wiley & Sons, Inc., 1994.

J. M. v. d. Mortel, J. E. Rooda, and N. J. M. v. d. Nieuwelaar, "Specification of a flexible manufacturing system using concurrent programming," *CERA International Journal on Concurrent Engineering: Research and Applications*, vol. 3, no. 3, pp. 187–194, 1995.

T. Murata, "Petri nets: Properties, analysis, and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

K. G. Murty, *Linear Programming*, Chichester: Wiley-Interscience, 1983.

N. J. M. v. d. Nieuwelaar, "Project plan: A framework for development of machine control systems," , 2000, available through URL `http://se.wtb.tue.nl/~bvdnieuw`.

N. J. M. v. d. Nieuwelaar, "Skip zeroing of IF at exposure side," , 2002, patent application ASML ref. P-0346, application number Europe: 03255923.9 (EP 1 404 712 A2), US: 10/665,351, China: 03125493.4, Japan: 2003-330017, Korea: 10-2003-0065530, Singapore: 200305621-5, Taiwan: 92125033.

N. J. M. v. d. Nieuwelaar, W. H. G. A. Koenen, J. Onvlee, H. P. J. v. Lierop, R. J. Dumont, M. A. R. Stoets, and J. E. Rooda, "Run-time conditional sequencing mechanism for supervisory machine control," , 2003a, patent application ASML ref. P-1704, application number Europe: 03256456.9, Japan: 2004-286595.

N. J. M. v. d. Nieuwelaar, J. M. v. d. Mortel-Fronczak, N. C. W. M. Braspenning, and J. E. Rooda, "Predictive scheduling in complex manufacturing machines: scheduling alternatives and algorithm," *submitted to IEEE TAC*.

N. J. M. v. d. Nieuwelaar, J. M. v. d. Mortel-Fronczak, N. C. W. M. Braspenning, and J. E. Rooda, "Predictive scheduling in complex manufacturing machines: machine-specific constraints," *submitted to IEEE TSM*.

N. J. M. v. d. Nieuwelaar, J. M. v. d. Mortel-Fronczak, and J. E. Rooda, "Design of supervisory machine control," in *Proceedings of the European Control Conference 2003*, K. Glover and J. Maciejowski, Eds., 2003b, cD-ROM.

N. J. M. v. d. Nieuwelaar, J. Onvlee, and R. Boumen, "Synchronization of wafer scanner and track to increase lithocell productivity," , 2004a, patent application ASML ref. P-1818, application number US: 10/781,945.

N. J. M. v. d. Nieuwelaar, J. Onvlee, H. P. J. v. Lierop, R. Boumen, R. J. Dumont, and J. E. Rooda, "Supervisory machine control featuring dynamic scheduling," , 2004b, patent application ASML ref. P-1885, application number US: 10/852,678.

N. J. M. v. d. Nieuwelaar, J. Onvlee, H. P. J. v. Lierop, N. C. W. M. Braspenning, J. Rooda, M. M. H. Driessen, J. F. Groote, M. Hendriks, and F. Vaandrager, "Run-time, model based supervisory control of manufacturing machines," , 2003c, patent application ASML ref. P-1784, application number US: 10/743,320.

D. Ouelhadj, P. I. Cowling, and S. Petrovic, "Utility and stability measures for agent-based dynamic scheduling of steel continuous casting," in *IEEE International Conference on Robotics & Automation*, 2003, pp. 175–180.

D. M. R. Park, "Concurrency and automata on infinite sequences," in *Proceedings of the 5th GI-Conference*, P. Deussen, Ed., no. 104 in LNCS, Springer–Verlag, 1981, pp. 167–183.

J. Park and S. A. Reveliotis, "Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings," *IEEE Transactions on Automatic Control*, vol. 46, no. 10, pp. 1572–1583, 2001.

M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, 1995.

R. G. Qiu and S. B. Joshi, "A structured adaptive supervisory control methodology for modeling the control of a discrete event manufacturing system," *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol. 29, no. 6, pp. 573–586, 1999.

P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

S. E. Ramaswamy and S. B. Joshi, "Deadlock-free schedules for automated manufacturing workstations," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 3, pp. 391–400, 1996.

A. Ramirez-Serrano and B. Benhabib, "Supervisory control of flexible-manufacturing workcells that allow the production of a priori unplanned part types," in *IEEE International Conference of Systems, Man and Cybernetics*, 2000, pp. 2127–2131.

J. Renders, E. Rossignol, M. Becquet, and R. Hanus, "Kinematic calibration and geometrical parameter identification for robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 6, pp. 721–732, 1991.

S. A. Reveliotis, M. Lawley, and P. Ferreira, "Polynomial-complexity deadlock avoidance policies for sequential resource allocation systems," *IEEE Transactions on Automatic Control*, vol. 42, no. 10, pp. 1344–1357, 1997.

S. Rostami and B. Hamidzadeh, "Optimal scheduling techniques for cluster tools with process-module and transport-module residency contraints," *IEEE Transactions on Semiconductor Manufacturing*, vol. 15, no. 3, pp. 341–349, 2002.

E. Roszkowska, "Supervisory control for deadlock avoidance in compound processes," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 34, no. 1, pp. 52–64, 2004.

J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Guide*, Reading: Addison Wesley, 1999.

J. Ryu and A. Rauf, "A new method for fully autonomous calibration of parallel manipulators using a constraint link," in *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics Proceedings*, 2001, pp. 8–12.

I. Sabuncuoglu and M. Bayiz, "Analysis of reactive scheduling problems in a job shop environment," *European journal of operational research*, vol. 126, pp. 567–586, 2000.

SEMI, "Standard for definition and measurement of equipment productivity," Tech. rep., Semiconductor Equipment an Materials International, 2000, available through URL `http://www.semi.org/`.

Y. Shin, T. Lee, J. Kim, and H. Lee, "Modeling and implementing a real-time scheduler for dual-armed cluster tools," *Computers in Industry*, , no. 45, pp. 13–27, 2001.

S. F. Smith, "Is scheduling a solved problem?" in *Multidisciplinary International Conference on Scheduling : Theory and Applications(MISTA'03)*, G. Kendall, E. Burke, and S. Petrovic, Eds., ASAP, University of Nottingham, UK, 2003, pp. 11–20.

W. Stallings, *Operating Systems – Internals and Design Principles*, Prentice–Hall, 1998.

J. A. Starzyk, J. Pang, S. Manetti, M. C. Piccirilli, and G. Fedi, "Finding ambiguity groups in low testability analog circuits," *IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications*, vol. 47, no. 8, pp. 1125–1137, 2000.

E. Szelke and R. M. Kerr, "Knowledge-based reactive scheduling," in *Proceedings of the IFIP TC5/WG5.7 international workshop*, 1993.

P. Toth and D. Vigo, *Predictive control with constraints*, Philadelphia: SIAM, 2002.

G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing systems: a framework of strategies, policies and methods," *Journal of scheduling*, vol. 6, no. 1, pp. 35–58, 2003.

G. X. Viennot, "Heaps of Pieces, I: Basic definitions and combinatorial lemmas," in *Combinatoire Enumerative*, G. Labelle and P. Leroux, Eds., pp. 321–350, New York: Springer, 1986.

N. Viswanadham, Y. Narahari, and T. L. Johnson, "Deadlock prevention and deadlock avoidance in flexible manufacturing systems using petri net models," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 6, pp. 713–723, 1990.

C. W. Wampler, J. M. Hollerbach, and T. Arai, "An implicit loop method for kinematic calibration and its application to closed-chain mechanisms," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 5, pp. 710–724, 1995.

Y. Wang and Z. Wu, "Deadlock avoidance control synthesis in manufacturing systems using model checking," in *IEEE American Control Conference*, vol. 2, 2003, pp. 1702–1704.

M. Wennink, "Algorithmic support for automated planning boards," Ph.D. dissertation, Eindhoven University of Technology, The Netherlands, 1995.

D. H. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

H. H. Wu and R. K. Li, "A new rescheduling method for computer based scheduling systems," *International journal of production research*, vol. 33, no. 8, pp. 2097–2110, 1995.

R. A. Wysk, N. S. Yang, and S. Joshi, "Detection of deadlocks in flexible manufacturing cells," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 6, pp. 853–859, 1991.

G. Yang and I. M. Chen, "Kinematic calibration of modular reconfigurable robots," *Journal of Robotics Systems*, vol. 16, no. 4, pp. 213–225, 1999.

H. J. Yoon and D. Y. Lee, "Deadlock-free scheduling of photolithography equipment in semiconductor fabrication," *IEEE Transactions on Semiconductor Manufacturing*, vol. 17, no. 1, pp. 42–54, 2004.

S. Yovine, "KRONOS: a verification tool for real-time systems," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1/2, pp. 123–133, 1997.

M. Zweben and M. S. Fox, *Intelligent scheduling*, San Francisco: Morgan Kaufmann, 1994.

Samenvatting

Dit proefschrift gaat over supervisory control van complexe productiemachines. ASML wafer scanners zijn de carriers van dit onderzoek. Een wafer scanner is een representatief voorbeeld van een complexe productiemachine, bestaande uit vele mechatronische systemen. In een complexe productiemachine bestaan vele mogelijkheden om de beschikbare middelen (resources) aan te wenden voor het uitvoeren van taken (tasks) die leiden tot het gewenste productiedoel, resulterend in verschillend gedrag van de machine. Supervisory Machine Control (SMC) is verantwoordelijk voor de bepaling van wanneer welke taken gedaan worden door welke middelen.

Het doel van dit project is om een geschikte formele methode te ontwikkelen voor de specificatie van supervisory control van complexe productiemachines. Er moet rekening gehouden worden met lastige eisen aan SMC van complexe productiemachines. Ten eerste zijn de productietaken in grote mate afhankelijk van het productrecept, waarvoor SMC flexibel moet zijn. Dit betekent dat SMC een stroom van gemixte producttypes moet kunnen verwerken, die gelijktijdig worden geproduceerd. Ten tweede moet SMC het gedrag van de machine kunnen optimaliseren door haar middelen op een zo goed mogelijke manier in te zetten binnen haar productiebeperkingen. Wat het beste is kan afhangen van de karakteristieken van het recept. Ten derde moet SMC passen in de dynamische omgeving waar zij in zit. Dit betekent dat zij op allerlei signalen vanuit haar omgeving moet reageren zonder onnodig besturingsoverhead te introduceren. Een heel belangrijk signaal is het falen van een taak: een exceptie, die vraagt om een herstelreactie van SMC om tussenkomst van een mens te voorkomen. Tenslotte moet SMC zich lenen om eenvoudig functionaliteit toe te voegen en te wijzigen, om de stijgende ontwikkelsnelheid in de industrie bij te kunnen houden.

Om de bovengenoemde eisen te vervullen is een scheduling gebaseerd SMC concept ontwikkeld. Om de te nemen besturingsbeslissingen te structureren wordt een gelaagd task-resource raamwerk gebruikt. Vanuit het oogpunt van SMC kan een machine beschouwd worden als een 'task resource' systeem (TRS). Taken kunnen geassocieerd worden met productieprocessen, terwijl resources geassocieerd kunnen worden met mechatronische systemen. Het transformeren van een productie-opdracht in machinegedrag kan gestructureerd worden in drie fases, gedurende welke rekening gehouden dient te worden met de beperkingen van de machine. Eerst moet er een scheduling probleem geïnstantieerd worden voor de productie-opdracht. Deze transformatie wordt *instantiating* genoemd. Vervolgens moeten middelen toegekend worden aan de taken in het geïnstantieerde scheduling probleem in een bepaalde volgorde, rekening houdend met het feit dat middelen slechts bepaalde taken uit kunnen voeren en maar één tegelijkertijd. Deze transformatie wordt *selecting* genoemd. De geselecteerde volgorde van taken die

uitgevoerd moeten worden door de geselecteerde middelen kunnen toestandsovergangen tussendoor tot gevolg hebben. Tenslotte kunnen start- en eindtijden toegekend worden aan de geselecteerde taken, rekening houdend met de duur van de taken. Deze transformatie wordt *timing* genoemd. De combinatie van de selecting en timing transformatie wordt ook wel scheduling genoemd. Gedurende de drie transformatiefases instantiating, selecting en timing, moeten keuzes gemaakt worden. De consequenties van een keuze in een bepaalde transformatiefase kan alleen bepaald worden door de opvolgende transformaties te doen. Vandaar dat een transformatiefase sterk leunt op informatie van opvolgende fases, wat aangegeven is in het gelaagde TRS raamwerk. In dit proefschrift worden de drie transformatiefases formeel gedefinieerd.

In dit project is een voorspellend-reactief SMC raamwerk ontwikkeld dat het gelaagde TRS raamwerk in zich heeft in de vorm van TRS translatiefuncties. Verschillende methoden of scenario's om te reageren op verschillende types van besturingsimpulsen worden beschreven middels deze translatiefuncties. Voor de 'goed weer' impulsen is het belangrijk om besturingsoverhead te voorkomen. Dit wordt gedaan door zeker te stellen dat reactie - indien mogelijk - plaatsvindt parallel aan de productieprocessen. Voor besturingsimpulsen die excepties betreffen is het belangrijker om op een robuuste manier te herstellen dan om besturingsoverhead te voorkomen. Vandaar dat de reactie op excepties sequentieel is. Bij het herstellen van excepties worden dezelfde transformatiefuncties gebruikt als bij de bovengenoemde 'goed weer' impulsen, wat een elegante eigenschap is.

De schedulingtransformatie die in SMC zit gebruikt heuristische filters om snel een goed schedule te vinden. Er zijn verschillende aanpakken ontwikkeld om het vastlopen (deadlock) van de machine te voorkomen. Bij één van hen wordt een model checker gebruikt om een filter te configureren dat deadlock voorkomt, terwijl het aantal schedules zo min mogelijk wordt beperkt. Een specifieke verificatie-aanpak is ontwikkeld om zeker te stellen dat het ontwerp van de filterconfiguratie inderdaad niet kan resulteren in ongeldig machinegedrag zoals deadlock. Deze aanpak maakt gebruik van de specifieke structuur van het schedulingmodel om toestandsruimte-reductietechnieken toe te passen. Deze technieken maken het mogelijk om gevallen van praktische omvang te verifiëren.

Er zijn twee instantiatie-aanpakken ontwikkeld. De eerste maakt gebruik van een database van instantiatieregels en bouwblokken om een schedulingprobleem te genereren die de productie-opdracht vervult. De andere aanpak gebruikt metataken en hun pre- en postcondities om te zoeken naar instanties van schedulingproblemen die de productie-opdracht vervullen, zoals in game theory. Deze aanpak is tevens uitgewerkt voor kinematische kalibraties. Kinematische kalibratie van precisiemachines valt onder SMC omdat afwijkingen en drifteffecten in de hardware gecorrigeerd moeten worden tijdens productie.

Toepassing van het voorgestelde besturingsconcept heeft significante voordelen vergeleken met de huidige gang van zaken bij ASML. De verwachting is dat de SMC gerelateerde software ontwikkelingsinspanning reduceert met een factor 2, en dat de machineprestatie verbetert met orde-grootte 1 %. Een beperkte versie van het besturingsconcept is succesvol geïmplementeerd in een deel van de besturingssoftware van het ASML TWINSCAN besturingsplatform. Een road-map die het evolutionair uitrollen van het concept in de komende jaren behelst is in ontwikkeling.

Curriculum Vitae

N.J.M. (Barend) van den Nieuwelaar was born on June the 6th, 1972 in Tilburg, the Netherlands. After finishing Atheneum B at the Theresia Lyceum in Tilburg in 1990, he started his studies at the Eindhoven University of Technology, Department of Mechanical Engineering. During the internship he co-authored a journal article [J.M. van de Mortel et al., 1995]. He carried out his final project at Schelde Energy and Environmental Systems in Vlissingen, where he studied the assembly of steam equipment in an electricity generation plant. He graduated in May 1996.

From June 1996 till March 1997 he worked at Schelde Energy and Environmental Systems as project planner and cost controller. In April 1997 he started working at Steelweld BV, where he has set up a project planning and cost control team to support project leaders by supplying project progress and forecast information. He was leader of this team and advised the management team.

In January 2001, he started working at ASML as a member of the software architecture group. From then, he has worked on a framework for the development of machine control systems, resulting in this PhD thesis.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Scheduler Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences.* Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes.* Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes.* Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking.* Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols.* Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents.* Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ.* Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking.* Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.* Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity.* Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing.* Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing.* Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining.* Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL.* Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand.* Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks.* Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing.* Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions.* Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components.* Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components.* Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method.* Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving.* Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems.* Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation.* Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments.* Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications.* Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications.* Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems.* Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs.* Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell.* Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation.* Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets.* Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics.* Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station.* Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents.* Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures.* Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling.* Faculty of Mechanical Engineering, TU/e. 2004-21