Multi-layer system modelling and verification of fine wafer alignment

Leemans, M.

Award date:
2014

Department of Mathematics and Computer Science
Formal System Analysis
Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands

Multi-layer system modelling and verification of fine wafer alignment

ASML Graduation Project (Public version)

Author
Maikel Leemans
m.leemans@student.tue.nl

University Supervisor
Jan Friso Groote
j.f.groote@tue.nl

Company Supervisors
Kees Kotterink
kees.kotterink@asml.com
Sven Weber
sven.weber@asml.com
Wouter Tabingh Suermondt
wouter.tabingh.
suermondt@asml.com

Order issuer
ASML

Date
August 22, 2014

Version
1.0
Public version

Where innovation starts

Contents

Part I

Case study introduction

1 Introduction

In this chapter, we state the goal of the assignment, describe the assignment case study and domain, state the assignment objectives and provide an outline of the rest of this document.

1.1 Assignment goal

The goal of this assignment is to *investigate academic techniques* to get insight into the (discrete event) *system behaviour* of the *ASML TWINSCAN system*. This is done by applying these techniques to the *(fine) wafer alignment functionality case study*. We will gain insight by establishing an unambiguous, correct description of this system behaviour. Additionally, we will provide feedback and advice on the system architecture.

In this assignment, parts of a complex industrial system were quickly mastered and reduced to its core concepts, using academic methodologies.

1.2 Preliminaries

The basic notions given below are used throughout this report.

(System) behaviour
Behaviour is anything that an organism or system does, involving action and response to stimulation.

Interface
An interface is a common boundary or interconnection between systems, where interaction or communication is achieved.

Requirement
A requirement specifies what behaviour a system/process should (needed functionality) and should not have (safety constraints).

Process
A process is a control mechanism, which defines a series of actions that lead to a particular result.

mCRL2
mCRL2 (micro Common Representation Language, version 2 [2, 6]) is a behaviour specification language and toolset for modelling and analysing communicating systems.

1.3 The ASML NXT3 TWINSCAN wafer scanner

1.3.1 The ASML lithographic machine

The ASML TWINSCAN lithographic machine (hereafter "the system") is used to manufacture integrated circuits on a silicon *wafer*. The function of an ASML lithographic machine is to expose specific parts of a wafer with the right amount of light. Typically, a wafer enters the ASML machine multiple times. Each time a new layer is added to the integrated circuit structure being built on the wafer. This structure is created via the projection of a pattern or image onto the wafer. We create this pattern by using a light source and a reticle mask. See Figure 1.1 for a context sketch of the ASML wafer exposure in the lifetime of wafer development.

If the wafer is not exactly at the focus point of the projected light, we have a *focus error*. If the wafer is not exactly in the same space as the previous expose, the layers will not exactly stack, and we have an *overlay error*. Wafers are typically processed in batches. A batch of wafers is also called a *LOT*.

1.3.2 On performance factors, dual stages and scanning

The two key performance factors for ASML lithographic machines are *accuracy* and *throughput*. Accuracy is measured in terms of focus and overlay errors. Throughput is measured in terms of *wafers per hour* processed.

In order to improve accuracy, series of measurements are needed to know exactly where the wafer is. These measurements take time. In order to improve throughput, there are two physical stations in the TWINSCAN machine (see also Figure 1.2): one for measuring the wafer (the *Measurement Station*), one for exposing the wafer (the *Exposure Station*). This way, measuring the wafer can be done in parallel with exposing another wafer. Consequently, this increases the number of wafers processed per hour (i.e., throughput).

In order to reduce the cost and the size of the lenses involved, the *scanning* principle is used in the Twinscan machine. Scanning means that instead of exposing the complete image at once, the image is drawn on the wafer, see also Figure 1.4. This requires the reticle and wafer to move in sync. The same is true for the measurement station. In order to "read" reference points, which tells us where the wafer is, we need to coordinate the wafer movement and the sensor hardware utilization. In order to be able to move the wafer, the wafer is placed on a waferstage or chuck.

1.3.3 Physical overview

In the ASML TWINSCAN machine, wafers to be processed are loaded onto a *waferstage*. The waferstage or chuck is the component that carries the wafer, and moves the wafer

around in the machine. There are two chucks, one at the measurement station and one at the expose station, allowing two wafers to be processed at the same time.

The two chucks move wafers between the measurement and expose station (as is indicated in Figure 1.2). Spanning these two stations is the *Metroframe*. The Metroframe is a physical frame that houses the lenses for exposure, and sensors for measurement (see also Figure 1.3).

1.4 Case Study – Wafer Alignment

1.4.1 Life of a Wafer and the Measurement Sequence

Recall, in order to *expose a wafer* (projecting a reticle pattern onto the wafer) while minimizing focus and overlay errors, we need to know with great accuracy where the wafer is and how the wafer is deformed. That is, we need to determine a set of wafer parameters that tell us the position and deformation of the wafer. To obtain the required information, we need to perform a series of measurement steps. The order in which these steps are performed are captured in the *Measurement Sequence*, which is executed at the measurement station. See Figure 1.5 for a context sketch of the measurement sequence in the Life of a Wafer inside the system. The Life of a Wafer is the complete sequence of actions a wafer goes through, from the moment it enters the system till the moment it leaves the system.

Due to the nature of the system, the scale operated at and the sensors used, we measure the wafer indirectly. In the measurement sequence, we determine the position of the wafer with relation to a fixed point, which is the waferstage. In short, this means we have two main activities in a measurement sequence: measuring where the waferstage is, and measuring where the wafer is on the waferstage.

1.4.2 Measurement sensors

The two main types of sensors at the measurement side are the *Level* and *Alignment* sensors. With the Level sensor, we measure the height and tilt of the wafer surface using interferometers. Complementary, the Alignment sensor measures the position of alignment marks on the wafer. Technical note: In this case study, we will assume that the SMASH alignment sensor is used.

There are two subsystems for aligning the waferstage: SPM and the TIS plates. Firstly, *Stage Position Measurement (SPM)* assists with positioning the waferstage by providing sensor feedback during movements. The SPM sensors are attached to the Metroframe (see also Figure 1.3). Secondly, the *TIS plates* are a subsystem on top of the waferstage. A TIS plate consists of both marks and sensors. These marks are read using the alignment sensor at the measurement station. The sensors are used at the expose station to determine where the image will be projected on the wafer by determining the position of the reticle.

1.4.3 Case Study identification

To recap, the case study is to get insight into the behaviour of (fine[1]) wafer alignment measurement step, performed at the measurement station using the alignment sensors.

1.5 Objectives

The objectives of this assignment are:

- Identify and describe the wafer alignment functionality behaviour, its requirements and the domain concepts associated with the system behaviour, as identified in the case study.
- Model, analyse and verify the wafer alignment functionality behaviour in the behavioural specification language mCRL2.
- Reflect on the system behaviour, highlighting potential problems, and relating the work to alternative design approaches.

1.6 How to read this document

1.6.1 Document Identification

This document describes, analyses and reflects on the (fine) wafer alignment functionality in the ASML NXT3 TWINSCAN. Furthermore, this document elaborates on the approach, methodology and technologies used. In addition, a reflection on the architecting process at ASML and a possible redesign is provided.

1.6.2 Stakeholders, and where do I find which information?

This document was written with two major stakeholders in mind: ASML system architects, and TU/e research staff. Therefore, not all parts of the document are equally relevant for each stakeholder. In the Outline section presented below, a brief summary of all the chapters is given as a guideline. To aid the reader, given below is a list of questions, and the chapters in which those questions are answered.

- *Which system was modelled?* See Chapters 1 and 2.
- *How was the system modelled? / Which techniques were used?* See Chapter 3.
- *What are the overal results and conclusions?* See Chapter 11.
- *Where can I find a detailed reference of what was modelled?* See Chapters 4, 5, 7 and 8, and Appendix F.
- *Where can I find a detailed discussion of the analysis results?* See Chapters 6, 9 and 10.
- *Where can I find and index/reference of all the terminology used?* See Appendix C.

[1] Fine Wafer Alignment is a specific step in the Measurement Sequence, and is discussed in more detail in Chapter 5 and Appendix A

1.6.3 Outline

In Part I, we start in Chapter 2 with an explanation of the key domain concepts related to the system behaviour of the (fine) wafer alignment functionality. In Chapter 3 we will discuss the approach and methodology employed in obtaining, modelling and analysis the system behaviour. In addition, in Chapter 4 we will address the requirements which we will verify on the behavioural model.

As will be motivated in Chapter 3, we divided the system behaviour in three parts. In Part II, we will address the software-only aspects of the system behaviour. In this part, we will describe the associated model in Chapter 5 and provide the accompanied behavioural analysis in Chapter 6. In Part III, we will address the software-hardware interaction aspects of the system behaviour. In this part, we will describe the associated models in Chapters 7 and 8, and provide the accompanied behavioural analysis in Chapter 9.

In Part IV, we wrap things up with a system-level analysis in Chapter 10. In Chapter 11 we will reflect on the system design and architecting process. Finally, in Chapter 12 we will briefly propose a redesign, and provide suggestions for future work.

Throughout this document a lot of domain and ASML specific abbreviations and concepts are used. In Appendix C there is a reference list of these abbreviations and concepts.

The system description is translated to a formal behavioural model, which is added in Appendix F. The analysis is based on the verifiable requirements described in Chapter 4, which are translated into the modal mu-calculus (see Appendix E). These requirements are verified for all conceivable scenarios.

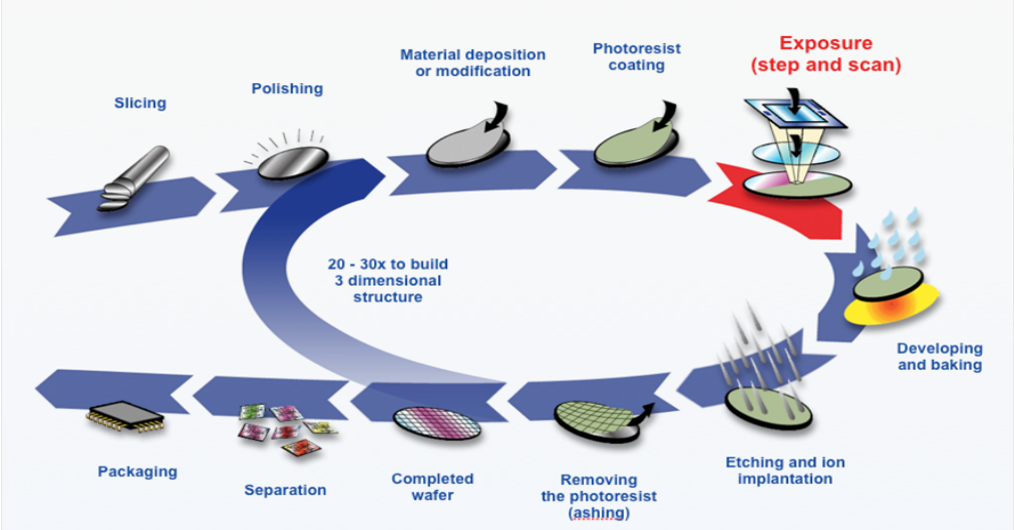

Figure 1.1: The wafer development cycle.

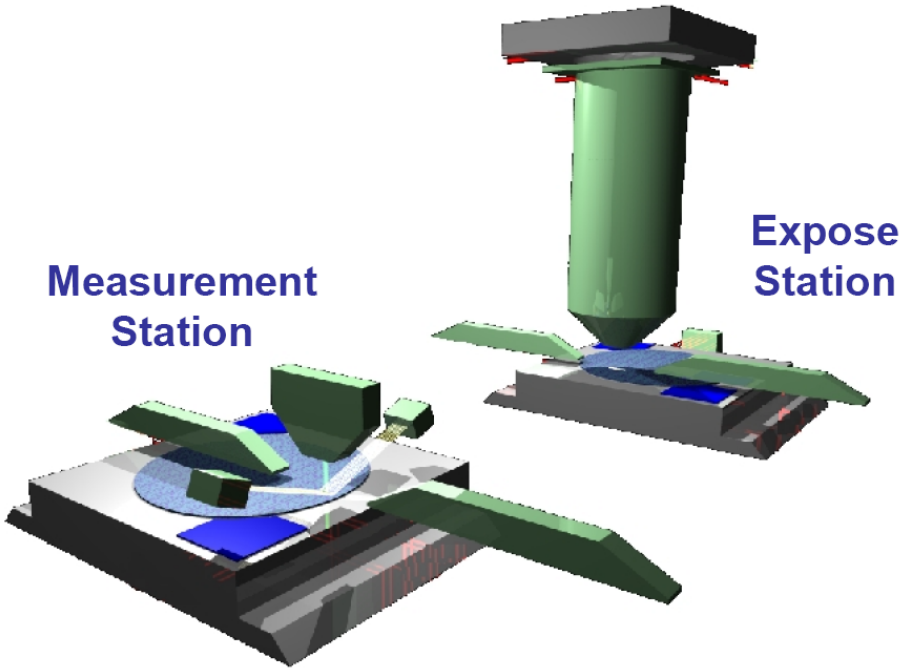

Figure 1.2: The measurement and expose stations within the ASML TWINSCAN machine.

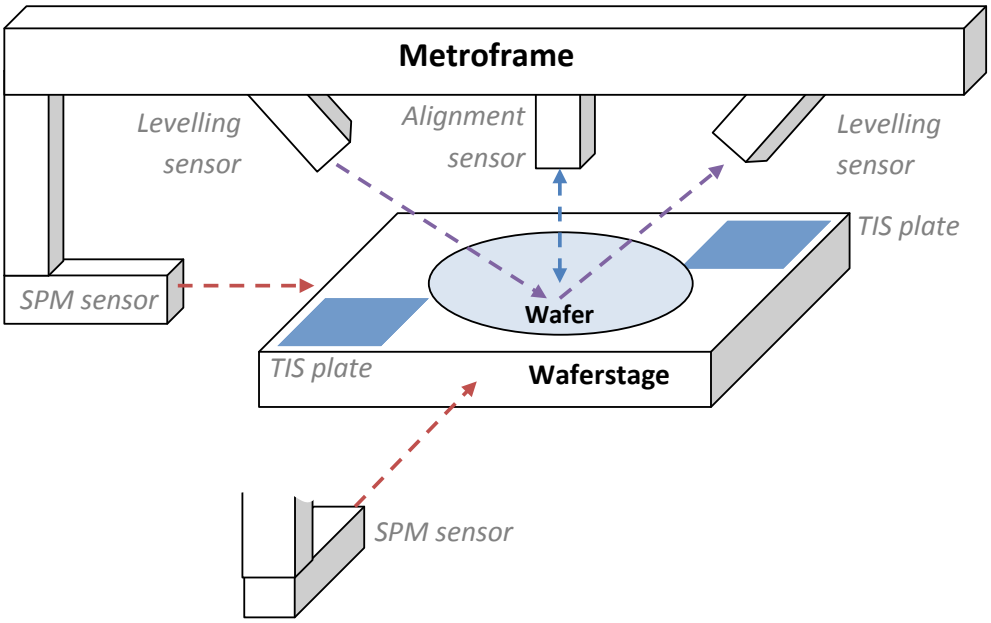

Figure 1.3: Overview of the physical world at the measurement station.

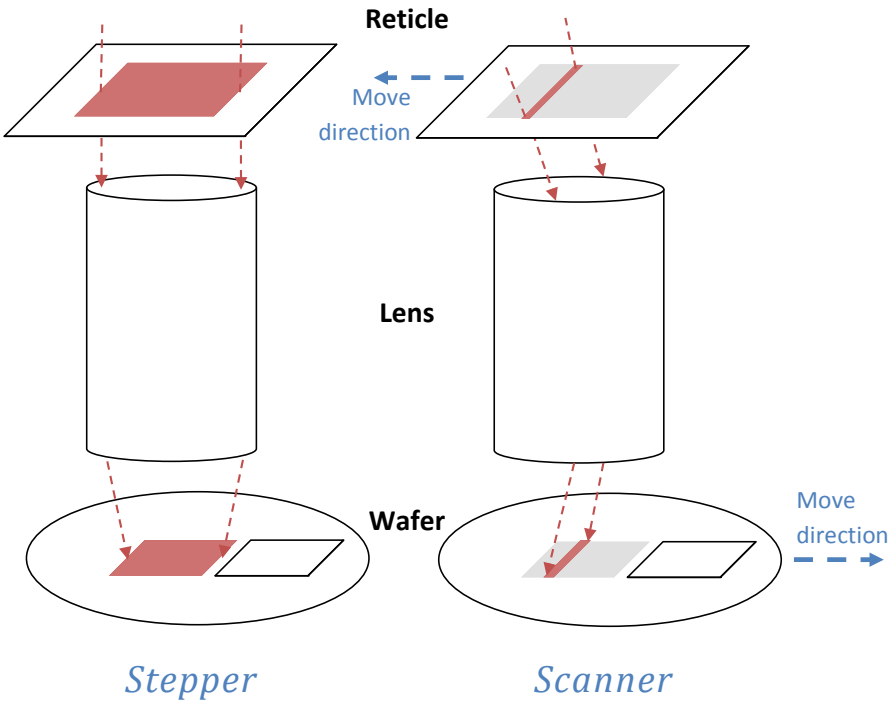

Figure 1.4: The difference between a one-time exposure (stepper behaviour), and a scan exposure (scanner behaviour).

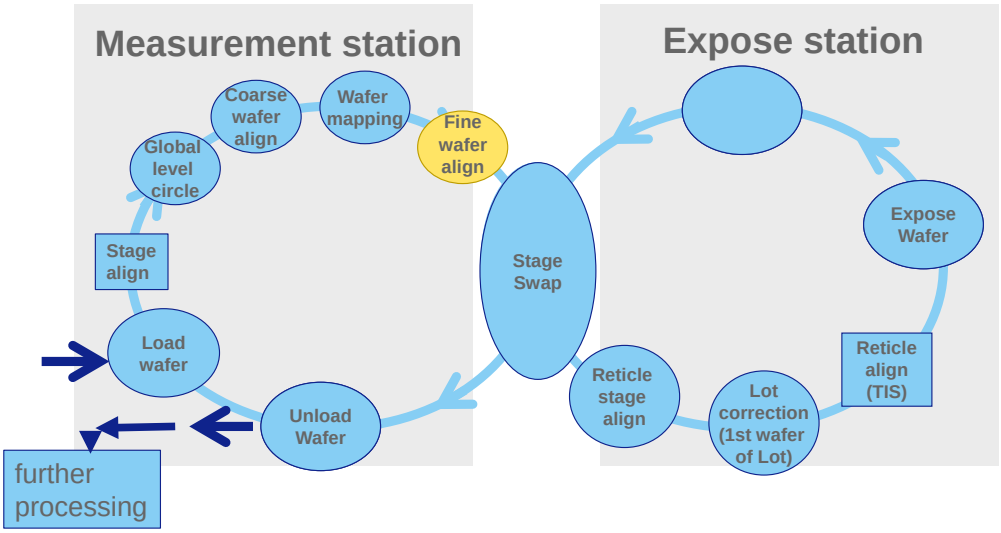

Figure 1.5: The life of a wafer in an ASML TWINSCAN machine.

2 Key domain concepts

In this chapter, we introduce and explain key domain concepts related to the system behaviour of the (fine) wafer alignment functionality. These concepts are needed to understand the modelled behaviour. In addition, we discuss the software architecture employed in the ASML TWINSCAN, and briefly address major protocols.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

3 Approach and Methodology

In this chapter, we will discuss the approach and methodology employed in obtaining, modelling and analysis the system behaviour. In addition, we will give a brief survey of related work and methodologies.

3.1 Related work

3.1.1 The mCRL2 language and toolset

The modelling is done in the behavioural specification language mCRL2 (micro Common Representation Language, version 2 [2, 6]). Recall, mCRL2 is a behaviour specification language and toolset for modelling and analysing communicating systems. For this analysis, we used mCRL2 release version 201310.0 [4].

The mCRL2 language and tookit, first released in 2003, is the successor to μCRL (developed in 1991). mCRL2 improved on μCRL by adding the basic datatypes as part of the language, where μCRL only contains a mechanism to define datatypes. mCRL2 builds upon the development work on process algebra's between 1970 and 1990 [6].

3.1.2 Related system analyses done in mCRL2

The mCRL2 toolset has previously been used for both high-level and low-level system analyses. It was shown that mCRL2 can be used for detailed modelling and analysing of existing low-level protocols. In [3], the FlexRay booting protocol was analysed using mCRL2, and in [10], the CAN state update protocol was investigated with the toolset. This type of analysis is generally used to verify key protocols and contracts on which larger systems are depending.

In [9] it was demonstrated that mCRL2 is able to validate large systems, by transforming large parts of the control software architecture deployed at the Compact Muon Solenoid (CMS) experiment at CERN. The formal models used in both analyses are of considerable size and complexity. In both cases, a thought-out decision was made regarding the detail-level of the modelled behaviour in order to avoid the state-space explosion problem.

A combination of high-level and low-level analysis was commissioned by the Dutch Ministry of Infrastructure and the Environment and documented in [11]. In this analysis, the software control stack of a movable bridge was verified in the mCRL2 toolset. In this case, a careful assessment was made regarding the detail-level of the behaviour at both

the low-level and high-level boundaries of the model. This was done in order to obtain a model of a manageable size that still covers both low-level and high-level concepts.

3.1.3 The need for formal specification

The need for formal and unambiguous descriptions was demonstrated in the experiment described in [7]. During this experiment, three senior architects were asked to design a lift control system based on an initial vague and ambiguous set of requirements. The varying approaches and resulting design demonstrated the effects of an informal design specification, which led the authors to the conclusion that "formal design representation languages" (like mCRL2) are needed.

3.1.4 Related formal analysis technologies

A toolset related to mCRL2 is Uppaal [1]. The toolset describe timed behaviour via networks of timed automata. Uppaal provides real-time and probabilistic model checking. In [15], both Uppaal and mCRL2 were used to verify models of a pacemaker. In this study, it turned out that Uppaal "was not suitable to deal with the full complexity of the software of the pacemaker" [13].

In [5], the SHE methodology and the POOSL formal specification language are discussed. The SHE methodology, or Software/Hardware Engineering methodology, is a framework for object-oriented specification and design of hardware/software systems. The SHE toolset uses the POOSL specification language. The POOSL language borrows traditional C-style programming language concepts in the form of asynchronous concurrent process objects for its formal specification. Like Uppaal, SHE focusses on real-time and probabilistic model checking.

The commercial tool ASD:Suite [14], developed by Verum, uses a formal software engineering approach called Analytical Software Design (ASD). Two types of formal models are employed in the ASD approach: design models and interface models, which are both described using state machines. Using the ASD:Suite, basic verifications like deadlock and illegal behaviour detection can be applied to the specified model. This model, which describes parallel communicating components or processes, can be translated into production code. In [8], this toolset is evaluated based on an industrial application within Philips Healthcare.

3.1.5 Related system analysis at ASML

The SHE methodology was previously used at ASML to analyse the synchronization protocols [12]. In this analysis, multiple subsystems and their associated state machines were added to the POOSL model. The model specification of these state machines was generated from an ASD specification.

3.2 System analysis methodology

A major challenge in obtaining the key concepts (Chapter 2) and formal descriptions (Chapters 5, 7 and 8) is the way in which information and knowledge is shared within ASML. Much of the design is documented in code and presentations, or shared verbally at the coffee machine.

As a result, the following (iterative) methodology was used during this project to reconstruct the key concepts and formal descriptions:

1. We started with identifying areas of interest. This was achieved by talking to domain experts, and looking up information in existing ASML documentation.
2. Based on the gathered information, key components and major parts of the overall behaviour were identified.
3. The obtained information was formalized as communicating processes, interfaces and in rare cases as requirements.
4. The processes and the associated communication were modelled in detail in the mCRL2 language. The act of creating formal models provided the first major feedback. Writing down the discovered ideas in a formal way allowed us to discover which parts we did not understand.
5. Using the mCRL2 toolset, the created mCRL2 models were analysed and verified.
6. Based on the performed analysis and verification, key concepts, ideas and problems were identified. These were discussed with domain experts, and, where needed, looked up in actual implementation code. Using the obtained insights and feedback, the model was adapted and corrected and alternative designs were investigated.

3.3 Modelling approach – The partial models approach

We are interested in both the low-level software/hardware interface protocols and high-level system behaviour, thereby covering all the layers of the software stack. As was also commented on in Section 3.1, we need to consider the detail-level of the model in order to prevent effects like state-space explosions.

The whole system behaviour is modelled by three partial models. A partial model describes a subset of the system, such that the boundaries of this subset are at the external interfaces. We "cut" the software architecture into partial models for two reasons. Firstly, the partial models reduce the overall complexity, avoiding state-space explosion problems. Secondly, the partial models allow us to analyse interfaces between conceptually different layers, which are found to be poorly understood.

Confidential. The information contained in this section is confidential, and has been removed from the public report.

3.4 Bridging application and modelling domains: Applied modelling patterns

In this section, we will elaborate on the way different behavioural aspects are captured in the formal models. Throughout the modelling process, several recurring patterns were identified and standardized in the mCRL2 models. We will highlight the major modelling patterns used in the mCRL2 models, which are included in Appendix F. Assumed is that the reader has basic knowledge of process algebra and the mCRL2 specification language.

The modelling patterns in this section bridge the gap between the application domain (the ASML system) and the modelling language mCRL2. The mCRL2 language is a generic behaviour specification language, and the capturing of the discussed domain concepts is a non-trivial task. The modelling patterns provide a "mental mapping" between the mCRL2 concepts and the application domain concepts. See Figure 3.1 for a depiction of how these domains relate to each other.

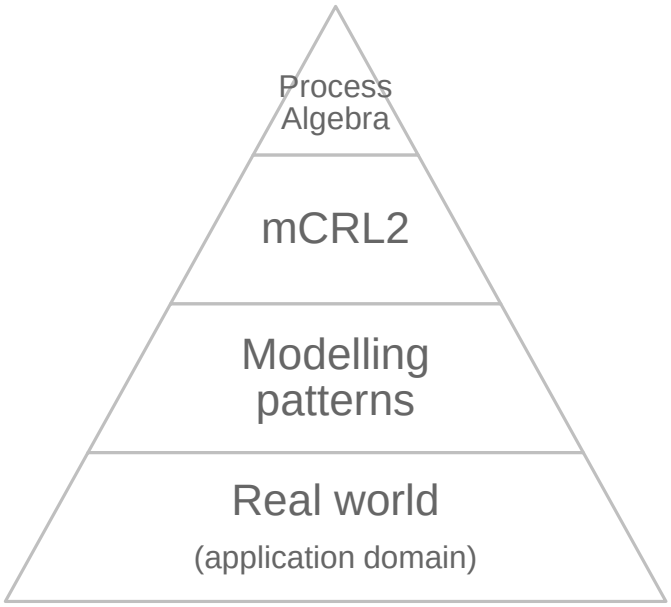

Figure 3.1: Illustration of the place of modelling patterns in the context of mCRL2 and the application domain (the ASML system).

3.4.1 Modelled processes and their relation to components

Two types of processes are modelled: component processes and network queue processes.

Component processes Each of the components described in Appendix B is modelled by its own process. This reflects the actual implementation used at ASML. For the subsystem components, the different sublayers involved each have their own processes. In Figure **??** an overview of the processes and their context is given.

Typically, we used process data to represent the state of the component, and in some cases, we introduced specialized state sorts. A component process definition begins with a choice of actions, each representing the receiving of a communicated message (e.g., an

interface call). When such a message is received, we perform the steps symbolizing the behaviour associated with the received message. After that, we recurse on the component process, with the modified state data.

Each component process has its own id. In the model we used the following sorts for enumerating the possible component ids:

```
% Component identifiers
sort CC_ID = struct
    CC_LO
  | ...
    ;
```

Network queue processes By defining separate network queue processes, we represented the indirect, delayed communication between components in the model. For more information on this type of communication and the *Message* sort, see the next section. The queue process is parameterized on the callee component id. Hence, we can reuse the process definition listed below. It can be instantiated for new components by passing along the new callee component id for the parameter *ccId*.

```
% Generic queue process used on XA interfaces for the receiving end
% Init: Queue_XA(ccId, RPC_Queue_Size, [])
proc Queue_XA(ccId: CC_ID, queueCapacity : Nat, queue: List(Message)) =
    % sending messages
    ((#queue < queueCapacity) -> (
        sum msg : Message .
        (targetId(msg) == ccId) ->
            q_enque(msg)
          . Queue_XA(queue = queue <| msg)
    ))

    % receiving messages
  + ((#queue > 0) -> (
        q_deque(head(queue))
      . Queue_XA(queue = tail(queue))
    ))
    ;
```

3.4.2 Communication between components

A generic approach is used to implement the interface concepts (see also Section **??** and Section **??**).

We distinguish two types of interface calls:

- *External Interfaces.* Communication between two parallel processes, the communication messages passed along are queueable. This modelling pattern covers both the function and event interfaces.
- *Internal Interfaces.* Direct messages-less communication that cannot be delayed. The Hardware/Software I/O communication was modelled with this type of interfaces.

External Interfaces For the External interfaces a simply queue process is placed between the two processes to facilitate the indirect, delayed communication. Recall that each component has two network queues:

- *Default queue.* Stores received function invoke and event messages.
- *Reply queue.* Stores received function reply messages.

External interfaces are identified by an interface id. In the model, we used the following sort for enumerating the possible interface ids:

```
% [XA] external interface sort.
% This structured sort represent the actual function calls and event triggers
sort XA_ID = struct
  | KMXA_lot_start(data: LOT_data)  | KMXA_R_lot_start
  | ...
  ;
```

Components communicate with each other via messages. The *message* type consists of an external interface identifier, and a source and destination component identifier:

```
sort Message = struct msg(sourceId: CC_ID, targetId: CC_ID, interfaceId: XA_ID);
```

There are three types of messages: function invoke, function reply and event messages (see also Section **??**). Invoking a function blocks the calling process until the return message (reply) is received. Event interfaces are modelled via a single external interface identifier.

External interface communication is modelled via the following actions:

```
% Send/Receive for usage of external interfaces
act snd, q_enque, XA_call,  % snd | q_enque -> XA_call      % Issue function call
    rec, q_deque, XA_accept % rec | q_deque -> XA_accept    % Process function
        call
    : Message;
```

Internal Interfaces For the Internal interfaces the two processes are directly involved in the interface interaction.

Internal interfaces are also identified by an interface id. In the model we used the following sort for enumerating the possible interface ids:

```
% [XI] internal interface sort.
sort XI_ID = struct
    InterfaceA
  | InterfaceB
  | ...
  ;
```

Internal interface communication is modelled via the following actions:

```
% Call/Return for usage of internal (library) interfaces
act call, invoke, XI_call,  % call | invoke -> XI_call      % Invoke method
    ret, return, XI_return  % ret | return -> XI_return     % Return method
    : XI_ID;
```

3.4.3 Illegal states

Another useful technique employed is the identification of states that should not be reachable. A typical example application is dequeuing from an empty queue. These states are referred to as illegal states.

In the mCRL2 model, we can identify the transitions that lead to an illegal state by carefully considering the data used in processes and covering all cases in the usage of this data. We marked these transitions with the special *error* action, followed by the *delta* action[1]. By providing additional data with this error action, we can quickly identify what goes wrong in the cases where we perform an error transition. In mCRL2 we used the following code to realise this idea:

```
act error : Act_Error_Codes;
% Specific information codes for error action, aiding in debugging model
sort Act_Error_Codes = struct
    ERROR_NotImplemented

  | WP_Dequeue__EmptyQueue
  | ...
  ;
```

3.4.4 Focussing on interface groups, using action renaming

We wish to analyse different aspects of the modelled system behaviour. To this end we employed the pattern described below, which allows us to indicate the focus or view of a state space without removing behaviour from the model. Using action renaming on the LPS[2], we can hide specific interface messages by renaming them to tau-steps. After applying branching bisimulation on the resulting linear transition system, we get a state space 'with the indicated focus'. This way, behaviour can be inspected and discussed with domain experts using uncluttered state space visualizations.

For this pattern we identified specific groups of interfaces, and specified two mappings for each group. The first mapping indicates (toggles) if we want to 'see' the group. If we do not want to see a group, the actions corresponding to that group are renamed to tau-steps. The second mapping identifies all the actions that correspond to the group.

As an example, suppose we only want to see the external interface interactions (i.e., only those involving the component KD) in the partial model *Logical Action Layer and Synchronization Control*. The mapping for toggling this focus group looks like:

```
map HideInternalComm : Bool;
eqn HideInternalComm = false; % true or false for turning the group on or off
```

Next, we specify the mapping that identifies all the action that correspond to this group:

```
map isInternalXA : Message -> Bool;
var msg : Message;
eqn isInternalXA(msg) = !(sourceId(msg) == CC_KD || targetId(msg) == CC_KD);
```

Finally, we specify the action rename rules as follows:

```
var msg : Message;
rename
    (HideInternalComm && isInternalXA(msg)) -> XA_call(msg)   => tau;
    (HideInternalComm && isInternalXA(msg)) -> XA_accept(msg) => tau;
```

[1] In mCRL2, the delta action symbolises a deadlock

[2] Linear Process Specification, the intermediate format used by mCRL2 in transforming a model specification into a linear transition system.

4 Requirements on the system and interface boundaries

This chapter defines requirements on the behaviour of the system. The goal of requirements formulation and verification is to better understand the system under analysis. Therefore, in Chapters 5, 7 and 8 models of the system are provided on which these requirements are checked.

Unfortunately, as far as known at the moment of writing, there is no verifiable set of requirements available within ASML. The list of requirements presented here is constructed based on the interviews with domain experts. The listed requirements are focused both at system level and at the boundaries between the software architectural layers. At the moment of writing, these areas are found to be the least understood. It should be remarked that these requirements are by no means complete, but are presented as an example in the right direction.

Every requirement in this chapter is translated into a modal μ-calculus formula. Appendix E lists the translated requirements, which are traceable via the provided identifiers. The modal μ-calculus formula is checked against the models described in Chapters 5, 7 and 8 and verified using mCRL2 (see Appendix F).

In this chapter, we distinguish three types of requirements:

Liveness requirements Liveness requirements enforce that certain interface calls will be enabled.

Functional requirements Functional requirements enforce that interface calls have certain effects.

Safety requirements Safety requirements enforce an order on activities in order to guarantee safe machine operations.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

Part II

Behavioural model – Software

5 Partial model description – Measure Control and Metrology

This chapter describes the processes and their interfaces included in the formal model corresponding to the partial model *Measure Control and Metrology*. Recall, this partial model is a software-only model, which focuses on the Measurement Sequence.

Where relevant for behavioural modelling, this chapter elaborates on the concepts from Chapter 2. The described model is modelled in mCRL2 (see [4]) and can be found in Appendix F. In Chapter 6, an analysis and verification based on this model is presented.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

6 Software – Analysis and verification

In this chapter, we will take a closer look at the system behaviour described by the partial model *Measure Control and Metrology*. Using the model as described in Chapter 5 and implemented in Appendix F, this Chapter provides the performed analysis and observations, and where applicable support the analysis with requirement verifications. In Chapter 11 we will reflect on the observations from this Chapter and present the lessons learned.

For the verification, the requirements in Chapter 4 are translated into μ-calculus formulae (see Appendix E), which were verified on the mCRL2 model (see Appendix F).

For the analysis, several configurations of the model are manually inspected using state-space visualizations and simulations, which were reviewed and discussed with domain experts.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

Part III

Behavioural model – Software/Hardware interaction

7 Partial model description – Logical Action Layer and Synchronization Control

This chapter describes the processes and their interfaces included in the formal model corresponding to the partial model *Logical Action Layer and Synchronization Control*. Recall, this partial model is a software-hardware model which focuses on the translation of measurement concepts to hardware interaction (i.e., from Logical Actions to Subsystem Actions).

Where relevant for behavioural modelling, this chapter elaborates on the concepts from Chapter 2. The described model is modelled in mCRL2 (see [4]) and can be found in Appendix F. In Chapter 9, an analysis and verification based on this model is presented.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

8 Partial model description – Synchronization Control and Subsystems

This chapter describes the processes and their interfaces included in the formal model corresponding to the partial model *Synchronization Control and Subsystems*. Recall, this partial model is a software-hardware model, which focuses on the synchronization protocol and actual hardware interactions.

Where relevant for behavioural modelling, this chapter elaborates on the concepts from Chapter 2. The described model is modelled in mCRL2 (see [4]) and can be found in Appendix F. In Chapter 9, an analysis and verification based on this model is presented.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

9 Software/hardware interaction – Analysis and verification

In this chapter, we will take a closer look at the system behaviour described by the partial models *Logical Action Layer and Synchronization Control* and *Synchronization Control and Subsystems*. Using the model as described in Chapters 7 and 8, and implemented in Appendix F, this Chapter provides the performed analysis and observations, and where applicable support the analysis with requirement verifications. In Chapter 11, we will reflect on the observations from this Chapter and present the lessons learned.

For the verification, the requirements in Chapter 4 are translated into μ-calculus formulae (see Appendix E), which were verified on the mCRL2 model (see Appendix F).

For the analysis, several configurations of the model are manually inspected using state-space visualizations and simulations, which were reviewed and discussed with domain experts.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

Part IV

Wrap up and conclusion

10 System-level analysis and verification

In Chapters 6 and 9 we took a closer look at each of the partial models and the behaviour modelled therein. In this chapter, we will zoom out and briefly look at the overall system behaviour.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

11 Results and conclusions

In this chapter, we will zoom out and look at what lessons we have learned from the presented formal system analysis. We reflect on both the system design and the observed architecting process that led to the discussed design.

We believe that with the results and conclusions (this Chapter), and the suggested redesign and follow-up analyses (Chapter 12), ASML can relatively easily apply the lessons learned in this report and effectively continue investigating their system architecture. In addition, we believe that the system overview presented in this report gives a more global overview of the ASML TWINSCAN system than is readily available within ASML.

11.1 Reflection on the system

In Chapters 6, 9 and 10 we provided the performed analysis and observations. In this section, we will reflect on the observations made, and present the lessons learned.

Confidential. The information contained in this section is confidential, and has been removed from the public report.

11.2 Reflection on the architecting process

In this section we will reflect on the architecting process at ASML as a whole. By looking at various aspects of the system and talking to different domain experts, we noted several approaches being employed by ASML employees. These approaches have had a notable effect on the system architecture.

11.2.1 Organizational structure around hardware influences system design

Thinking in physics and mechanics

At all the layers and levels at which the system can be described, ASML employees prefer to revert to the physical world, to the world they can touch and visualize. While this is not necessarily a bad habit, it can become a pitfall when one forgets about the higher-level ideas, original concepts and actual system behaviour.

One example result from this type of pitfalls is the observation in Section **??**: "the system is programmed upside down in terms of which concepts are used at which architectural level". As a result, concepts such as laser shutters end up in high-level architectural layers. Due to thinking in physics and mechanics, one is typically inclined not to consider the higher-level concepts that should be used instead.

Organization influences system design: Functional Clusters

Based on the physical world view on the system, low-level functionalities are identified and grouped together into Function Clusters (FCs). These FCs are organizational units that are tasked with addressing the challenges or problems in that specific domain. Typically, each FC ends up owning a set of components that are totally contained in that FC, and not shared with another FC. This includes software components. Hence, the result is that the software components are formed around the organization structure, which only considered the physical world to begin with. This approach tends to suffer from the same pitfall discussed in the previous section, where the higher-level conceptual overview is under-represented.

11.2.2 Problem solving approaches employed at ASML

Functional Clusters, knowledge sharing and the lack of system-level overview

Elaborating on the section about Functional Clusters (FCs), another effect of using these types of clusters is that architects (Functional Architects, or FAs) tend to consider only their cluster. The effects of this are that the persons responsible for developing and maintaining the overview limit their scope to their own cluster. As a result, at various clusters the same problems were discovered independently, and usually solved multiple times in very different ways. A good example of this is the multitude of Queuing Policies, as discussed in Section **??**.

It might be worthwhile to create disciplines that solely focus on a cross-FC behavioural aspect, such as the Measurement Sequence, without attaching it to a physical sensor. The idea of clusters should be revisited, and based on concepts and problems within the system as a whole, not a specific physical part.

Hot-fixes instead of going back to the original concepts, and the resulting documentation incompleteness

Due to the lack of overview, together with the complexity of the system in question, a lot of issues and bugs are found in late-stage integration and production, which need to be fixed quickly. Typically, these issues are fixed by the FC that owns the involved components, and are typically cured with hot-fixes. Considering the late stage in which these issues are discovered and the time-to-market pressure, this hot-fixing approach is understandable. However, a result is that the rethinking that is usually required is skipped.

As a result, a lot of 'lessons learned' moments seem to be skipped, and the original concepts slowly are disappearing and forgotten in the revised system design. Another

result from this lack of feedback and reviewing, the existing documentation is usually outdated as soon as it is released, and one avoids learning what needs to be documented in the first place. This is for example visible in the incomplete and incorrect interface contracts available within ASML.

By applying formal, model-based design and engineering techniques and promoting engineering by contract and early design validation & verification, many of these hot-fixes can be avoided. When applied correctly and consistently, these design principles yield fewer issues in late-stage integration and production, less hot-fixes, and better and more complete documentation.

12 Future work

In this chapter, we briefly discuss possible future work following up on the analysis presented in this report. We address three topics: a possible redesign, follow-up analyses that could improve the insight into the discussed system behaviour, and follow-up work to be done on the standardization of mCRL2 patterns.

12.1 Standardizing mCRL2 patterns

Throughout this case study, many recurring modelling patterns were identified during the development of the mCRL2 models. As was also documented in Chapter 3, several of these patterns can be standardized. Currently, the mCRL2 specification language feels like a powerful 'assembly-like' modelling language. Future work could focus on identifying and standardizing modelling methodologies, modelling patterns and best practices, much like was done in Chapter 3.

Such work would greatly lower the learning curve of using mCRL2, and provide new and existing users with many helpful guidelines and templates. Benefits that could result from this kind of standardization include:

- Providing users with tips in creating manageable models, which avoid state space explosion problems. Things to consider in this category include, amongst others, the trade-off between explicitly modelling intermediate (tau) steps and combining behaviour in multi-actions.
- Providing users with techniques for better understanding the modelled behaviour. For example, through applying the illegal state pattern, counterexample traces for requirement violations can be generated.
- Providing users with patterns that employ a conceptual mapping between behaviour modelling and implementation. Consider, for example, the representation of component external interfaces and the mapping to the underlying messages.

12.2 Analyses for future studies of the system

Confidential. The information contained in this section is confidential, and has been removed from the public report.

12.3 Proposed redesign

Based on the observations in Chapter 11, future work could focus on redesigning the system, incorporating the lessons learned. Ideally, such a redesign would promote the move towards a modular, plug-in software architecture with engineering by contract and early design validation & verification.

As an example, a simple redesign model was created. In this section, we will briefly highlights the innovations applied as a result of the lessons learned. In Appendix F the deadlock-free mCRL2 model corresponding to this redesign is given. In Figure **??** a high-level overview of the redesign is given.

Confidential. The information contained in this section is confidential, and has been removed from the public report.

Part V

Appendices

A Measurement Sequence Steps

To provide a bit of context, the measurement steps used in the behavioural models are described briefly below. Some of these steps are also illustrated in the high-level overview in Figure 1.5.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

B Software components involved in the case study

In this chapter, the software components involved with the wafer alignment case study are described. In addition, for each component the corresponding layer in the software architecture and the scope are given. Scope, in this context, means the components whose external interfaces are used by the indicated component.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

C Nomenclature

Accuracy Accuracy is a key performance factors for ASML lithographic machines, and is measured in terms of focus and overlay errors, page 8

Alignment sensor Sensor that measures the position of alignment marks on the wafer, page 9

Behaviour Behaviour is anything that an organism or system does, involving action and response to stimulation, page 7

Chuck Alias for waferstage, page 9

Exposure station Physical station in a TWINSCAN machine where the wafer is exposed, page 8

Focus error Error caused by the wafer being not exactly at the focus point of the projected light, page 8

Interface An interface is a common boundary or interconnection between systems, where interaction or communication is achieved, page 7

Level sensor Sensor that measures the height and tilt of the wafer surface using interferometers, page 9

Life of a Wafer The complete sequence of actions a wafer goes through, from the moment it enters the system till the moment it leaves the system., page 9

LOT A batch of wafers, page 8

mCRL2 mCRL2 (micro Common Representation Language, version 2 [2, 6]) is a behaviour specification language and toolset for modelling and analysing communicating systems, page 8

Measurement Sequence The sequence of actions, performed at the measurement station, to determine a set of wafer parameters that tell us where the wafer is, and how the wafer is deformed, page 9

Measurement station Physical station in a TWINSCAN machine where the wafer is measured prior to exposure, page 8

Metroframe A physical frame that houses the lenses for exposure and sensors for measurement, page 9

Overlay error Error caused by the wafer being not exactly in the same space as the previous expose, causing the layers to not stack exactly, page 8

Process A process is a control mechanism, which defines a series of actions that lead to a particular result, page 7

Requirement A requirement specifies what behaviour a system/process should (needed functionality) and should not have (safety constraints), page 7

Scanning Scanning means that instead of exposing the complete image at once, the image is drawn on the wafer, page 8

SMASH Smart Alignment Sensor Hybrid, page 9

SPM Stage Position Measurement, page 9

Stage Position Measurement System that takes care positioning the waferstage by providing sensor feedback during movements, page 9

Throughput Throughput is a key performance factors for ASML lithographic machines, and is measured in terms of wafers per hour, page 8

TIS plate Sensor and grating plate on the waferstage. The TIS sensor is mainly used for reticle align, page 9

Wafer A thin slice of semiconductor material, such as a silicon crystal, used in the fabrication of integrated circuits and other microdevices, page 8

Wafer exposure The act of projecting a reticle pattern onto the wafer, page 9

Waferstage The component that carries the wafer, and moves the wafer around in the machine, page 9

D Bibliography

[1] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal a tool suite for automatic verification of real-time systems. In Rajeev Alur, ThomasA. Henzinger, and EduardoD. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer Berlin Heidelberg, 1996.

[2] S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, J.W. Wesselink, and T.A.C. Willemse. An overview of the mCRL2 toolset and its recent advances. In N. Piterman and S.A. Smolka, editors, *Proceedings TACAS 2013*, number 7795, pages 199–213. Springer, 2013.

[3] Sjoerd Cranen. Model checking the flexray startup phase. In Marille Stoelinga and Ralf Pinger, editors, *Formal Methods for Industrial Critical Systems*, volume 7437 of *Lecture Notes in Computer Science*, pages 131–145. Springer Berlin Heidelberg, 2012.

[4] Technische Universiteit Eindhoven. mCRL2 Home. http://www.mcrl2.org/.

[5] M.C.W. Geilen, J.P.M. Voeten, P.H.A. van der Putten, L.J. van Bokhoven, and M.P.J. Stevens. Object-oriented modelling and specification using SHE. *Computer Languages*, 27(13):19 – 38, 2001. Visual Formal Methods-VFM'99 Symposium.

[6] J.F. Groote and M.R. Mousavi. *Modelling and Analysis of Communicating Systems*. MIT Press, 2014. To appear.

[7] Raymonde Guindon. Knowledge exploited by experts during software system design. *International Journal of Man-Machine Studies*, 33(3):279 – 304, 1990.

[8] Jozef Hooman, ArjanJ. Mooij, and Hans van Wezep. Early fault detection in industry using models at various abstraction levels. In John Derrick, Stefania Gnesi, Diego Latella, and Helen Treharne, editors, *Integrated Formal Methods*, volume 7321 of *Lecture Notes in Computer Science*, pages 268–282. Springer Berlin Heidelberg, 2012.

[9] Yi Ling Hwong, Jeroen J.A. Keiren, Vincent J.J. Kusters, Sander Leemans, and Tim A.C. Willemse. Formalising and analysing the control software of the compact muon solenoid experiment at the large hadron collider. *Science of Computer Programming*, 78(12):2435 – 2452, 2013. Special Section on International Software Product Line Conference 2010 and Fundamentals of Software Engineering (selected papers of FSEN 2011).

[10] M. Leemans and J.F. Groote. Formal system analysis of the Stella Solar Car. Technical report, Solar Team Eindhoven, Eindhoven University of Technology, The Netherlands, Juli 2013. confidential.

[11] S. Cranen S.E. Jurgens M. Leemans, R.P.J. Koolen and J.F. Groote. Analyse van besturingssystemen voor beweegbare bruggen (analysis of the control system for movable bridges). Technical report, Rijkswaterstaat, May 12 2014. confidential.

[12] Wouter Tabingh Suermondt. *FC-056 System Synchronization Model*. ASML.

[13] Technische Universiteit Eindhoven. mCRL2 website: Pacemaker. http://www.mcrl2.org/release/user_manual/showcases/Pacemaker.html. [Online, accessed 1 August 2014].

[14] Verum Software Technologies. ASD:Suite. http://www.verum.com/. [Online, accessed 1 ugust 2014].

[15] J. E. Wiggelinkhuizen. Feasibility of formal model checking in the Vitatron environment. Master's thesis, Technische Universiteit Eindhoven, 2007.

E Requirements in modal mu-calculus

In this appendix the translation to the model μ-calculus is given for every requirement in Chapter 4. This translation is the formalisation of the requirement, and these μ-calculus formulae are the actually verified descriptions.

Confidential. The information contained in this chapter is confidential, and has been removed from the public report.

F Setup and mCRL2 models used

In this chapter the actual mCRL2 models are listed, which are described in the Chapters 5, 7, 8 and 12.

F.1 Chapter overview

In section F.2, we will detail the hardware and software setup used. In addition, for the existing system, the models have been divided into three partial models:

Confidential. The information contained in this section is confidential, and has been removed from the public report.

F.2 Setup used

For the modelling, analysis, and verification, the following system configuration was used:

System Acer Aspire V3 - 771G

Processor Intel Core i7-3632QM 2.2GHz

Memory 8 GB DDR3 Memory

Operating system Windows 7 / 64-bit

mCRL2 version mCRL2 version 201310.0 / 64-bit

Environment Cygwin version 2.844 / 64-bit; with bash terminal

F.3 Existing system models

Confidential. The information contained in this section is confidential, and has been removed from the public report.

F.4 Redesign system model

Confidential. The information contained in this section is confidential, and has been removed from the public report.