Scalable model-driven software for stage alignment of ASML TWINSCAN machines: control and domain logic services

Citation for published version (APA):

Document status and date:
Published: 24/10/2018

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: 02. Aug. 2019
Scalable Model-Driven Software for Stage Alignment of ASML TWINSCAN Machines – Control and Domain Logic Services

Anita Asprovska
Scalable Model-Driven Software for Stage Alignment of ASML TWINSCAN Machines – Control and Domain Logic Services

Anita Asprovska

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

ASML

Steering Group

Lewis Binns (ASML)
Javad Vazifehdan (ASML)
Santiago Cifuentes Costa (ASML)
Anton Wijs (TU/e)

Date

October, 2018

Document Status

Public

PDEng report no.

2018/095

The design described in this report has been carried out in accordance with the TU/e Code of Scientific Conduct.
Abstract
Due to the accumulated technical depth of one of the software modules, which was accumulated during the last decade, ASML needs a complete redesign of the component using new technologies such as model-driven development. The design is required to follow the newly introduced architectural design pattern named DCA, which primary goal is to increase the modularity of the software, to improve the quality of the software and the work distribution between the engineers. This project is targeting the first step towards this goal of redesigning the complex component. The main focus of the project is the Control aspect (from DCA pattern) and the Domain Data Services. The final deliverable, according to the stakeholders, is considered as a satisfactory first step of the product roadmap.

Keywords
Model Driven Engineering, ASD:Suite, ASOME, domain modeling, control behavior, DCA, Separation of Concern.

Preferred reference

Partnership
This project was supported by Eindhoven University of Technology and ASML.

Disclaimer
Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the Eindhoven University of Technology or ASML. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Eindhoven University of Technology or ASML, and shall not be used for advertising or product endorsement purposes.

Disclaimer Liability
While every effort will be made to ensure that the information contained within this report is accurate and up to date, Eindhoven University of Technology makes no warranty, representation or undertaking whether expressed or implied, nor does it assume any legal liability, whether direct or indirect, or responsibility for the accuracy, completeness, or usefulness of any information.

Trademarks
Product and company names mentioned herein may be trademarks and/or service marks of their respective owners. We use these names without any particular endorsement or with the intent to infringe the copyright of the respective owners.

Copyright
Copyright © 2018. Eindhoven University of Technology. All rights reserved. No part of the material protected by this copyright notice may be reproduced, modified, or redistributed in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the Eindhoven University of Technology and ASML.
While many of the student projects held in ASML are focused on analysis and improvement of several aspects of our systems - quality metrics, complexity or automation- Anita’s project is targeted to be incorporated in our production code as full feature on it’s on. This represents a challenge for both, Anita and ASML.

On the last years an architectural design pattern is being introduced in our production software, named DCA, which intends to increase the modularity of our software, and improve work distribution, quality and effort to develop new features by the engineers. IVSA is an example of the need of such move, an existing feature that needs redesign due to its accumulated complexity.

The challenge for Anita has been to grasp our existing environment and feature design and translate that into a realizable and clear set of specifications and constrains. Along the way she has faced some of the realities of software design: the individual ideas and principles of the design are challenged when confronted with multiple specifications, requirements or just the sheer number of things to fulfill. Engineering is the art of finding the best compromise.

To get such compromise done it is crucial to find the main factors and Anita has shown a great deal of organization and ability to structure her task, learning our environment and, at the same time, challenging those aspects of our knowledge that are sometimes taken for the truth just because we have been exposed to them long enough.

Along the time Anita has been working on it, IVSA redesign importance has been recognized and desired for our next generation of machines where her work will serve as first step and guide towards a fully redesigned component. I have to personally thank Anita for her questions -and patience-, since they have helped me reflect over and clarify my own ideas and I can only hope she has also got a good experience with us as we had.

Project Supervisor
Santiago Cifuentes Costa
September 2018
Preface

This report is a summary of the “Scalable Model-Driven Software for Stage Alignment of ASML TWINSCAN Machines – Control and Domain Logic Services” project executed by Anita Asprovska, as part of her graduation assignment from the PDEng (Professional Doctorate in Engineering) program, offered by the Eindhoven University of Technology, Stan Ackermans Institute. The duration of the project was ten months and it was conducted under the supervision of ASML.

This report is primarily intended for readers with technical background in disciplines such as model-based technologies, domain specific languages, domain modeling, code generation and in general software engineering. However, no specialized knowledge in these disciplines is needed. Nevertheless, this report may also concern audience with non-technical background, such as management personnel.

The following table lists the chapters which might be interesting for different types of readers. Readers interested in the complete project are invited to read the entire report.

<table>
<thead>
<tr>
<th>Reader</th>
<th>Chapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software engineer</td>
<td>Problem Analysis</td>
</tr>
<tr>
<td></td>
<td>Domain Analysis</td>
</tr>
<tr>
<td></td>
<td>System Requirements</td>
</tr>
<tr>
<td></td>
<td>System Architecture</td>
</tr>
<tr>
<td></td>
<td>System Design</td>
</tr>
<tr>
<td></td>
<td>Implementation</td>
</tr>
<tr>
<td></td>
<td>Verification</td>
</tr>
<tr>
<td></td>
<td>Validation</td>
</tr>
<tr>
<td>Test engineer</td>
<td>Problem Analysis</td>
</tr>
<tr>
<td></td>
<td>Domain Analysis</td>
</tr>
<tr>
<td></td>
<td>System Requirements</td>
</tr>
<tr>
<td></td>
<td>Implementation</td>
</tr>
<tr>
<td></td>
<td>Verification</td>
</tr>
<tr>
<td></td>
<td>Validation</td>
</tr>
<tr>
<td>Project manager</td>
<td>Introduction</td>
</tr>
<tr>
<td></td>
<td>Problem Analysis</td>
</tr>
<tr>
<td></td>
<td>Stakeholder Analysis</td>
</tr>
<tr>
<td></td>
<td>Feasibility Analysis</td>
</tr>
<tr>
<td></td>
<td>Project Management</td>
</tr>
<tr>
<td></td>
<td>Retrospective</td>
</tr>
</tbody>
</table>

Anita Asprovska
October 2018
Acknowledgements

I owe the success of this project to the unprecedented help from all involved parties.

To begin with, I would like to thank Lewis Binns (Senior Software Architect) and Javad Vazifehdan (Senior Software Project Lead) for giving me the opportunity to carry out this project. I would like to thank you both for the continuous support, motivation and feedback throughout the project.

Next, my sincere gratitude goes to my company supervisor dr. Santiago Cifuentes Costa, who have invested his full effort in guiding me in the last ten months. I like to thank you for being a remarkable mentor by supporting me with your knowledge, patience and understanding. Your support was crucial for me and for the success of this project. I really enjoyed our discussions which thought me a lot about software engineering both as a craft and as a profession. I could not have asked for a better supervisor for my project.

Apart from my supervisor from ASML I would like to thank Edwin Jansen, Hugo Looijestijn, Arjan Holscher and all the people from the metrology department that provided support and interest in this project.

Furthermore, I wish to express my gratitude to my university supervisor dr. Anton Wijs. Your critical thinking, ideas and feedback encouraged me to explore the project in different views.

My gratitude goes to everybody involved in the PDEng program, especially Yanja Dajsuren, Ad Aerts, and Desiree van Oorschot for giving me the opportunity to be part of the program. Thank you for your support and guidance during these past two years, which helped me grow both professionally and personally. I would also like to thank my colleagues from OOTI generation 2016 for the great moments and experiences we shared together. To Ianislav Mintchev, my fellow colleague, than you for sharing the final graduation project experience with me.

Last but not least, I would like to thank my family, friends and boyfriend for their continuous support, patience and encouragement. Without their support this achievement would not have been possible, therefore I dedicate this milestone to them.

Anita Asprovska
October 2018
Executive Summary

In the process of developing software for a long period of time, it is inevitable to accumulate technical depth. The problem is that more than a decade’s worth of technical debt is hard to pay off. This causes yet more problems in extending or making any changes in the current software implementation. Such problem became a bottleneck for an existing module (IVSA in NXT systems) in ASML’s software. The module cannot be extended to implement an important new feature, which is currently one of the top ten issues at a major customer. Same problem is faced for the new generation of NXE machines.

Facing such issue, ASML software department decided to start with the development of the IVSA redesign using the new reference architecture (DCA) and employing model-driven engineering approach. The goal of the project is to make the first step towards the new design which intends to improve the modularity of the software as well as the work distribution. To meet this goal two parallel projects were conducted, this project focusing on the Control and Domain Logic Services and Project 2 focused on Algorithms and Domain Data Services.

After the analysis of the problem we choose, together with the trainee working on Project 2, to start with the development of the basic IVSA use case. This project delivered the design and implementation of the control logic integrated with the other two aspects. The deliverable from this project is intended to be incorporated in the production code after having the complete redesign (including all IVSA use cases).

This project serves as the first attempt of redesigning the module and a guide in the development of the new component which will replace the existing, complex implementation. The design chapter explains in details the design decisions and describes the design approach used in modeling the control behavior.

Each delivery of a feature was formally validated in order to make sure that the system is built correctly. Additionally, it was also verified by creating test cases to ensure the correct behavior is being implemented. It is recommended that the process of validation and verification is followed in the development of the next features, since it was proven that these processes are reliable for delivering high quality software. However, the current testing did not include integration testing which is highly recommended before the result of this project is incorporated in the production code.
# Table of Contents

Foreword ................................................................. i  
Preface ......................................................................... iii  
Acknowledgements ........................................................ v  
Executive Summary ........................................................ vii  
Table of Contents ........................................................... ix  
List of Figures .................................................................. xiii  
List of Tables ................................................................... xv  
Abbreviations ................................................................... xvii  

1. Introduction ............................................................... 1  
   1.1 Context .................................................................... 1  
      1.1.1. ASML metrology department .......................... 1  
   1.2 Outline ...................................................................... 1  

2. Problem Analysis ......................................................... 3  
   2.1 Context .................................................................... 3  
   2.2 Project description .................................................. 4  
      2.2.1. Project goal .................................................. 4  
      2.2.2. Project scope .............................................. 4  
   2.3 Roadmaps ............................................................... 4  
      2.3.1. Business Roadmap ..................................... 4  
      2.3.2. Technology Roadmap ................................. 4  
      2.3.3. Product Roadmap .................................... 5  
   2.4 Design opportunities ............................................. 5  
      2.4.1. Methodical approach ................................. 5  
      2.4.2. Functionality ........................................... 5  
      2.4.3. Complexity ............................................... 5  

3. Domain Analysis ........................................................ 6  
   3.1 Reference Architecture ......................................... 6  
      3.1.1. Data ......................................................... 7  
      3.1.2. Control .................................................... 8  
      3.1.3. Algorithm ............................................... 10  

4. Stakeholder Analysis .................................................... 13  
   4.1 Introduction .......................................................... 13  
   4.2 Stakeholders .......................................................... 14  

5. Feasibility Analysis ....................................................... 19
5.1 Risks .....................................................................................19

6. System Requirements ................................................................23
  6.1 Introduction ...........................................................................23

7. System Architecture ..................................................................25
  7.1 Introduction ...........................................................................25
  7.2 Application dimension .............................................................26

8. System Design ...........................................................................27
  8.1 Construction dimension ............................................................27
  8.2 Design decisions .................................................................28
     8.2.1. Rationale for choosing DCA pattern ..........................28
     8.2.2. Rationale for choosing ASD .................................29
     8.2.3. Following SOLID principles ....................................31
  8.3 Control design .......................................................................32
     8.3.1. Domain model ...........................................................32
  8.4 Failure handling .................................................................32

9. Implementation ..........................................................................35
  9.1 Code Generation ....................................................................35
  9.2 Glue code ..............................................................................35

10. Verification ...............................................................................37
  10.1 Introduction ..........................................................................37
  10.2 Verification of IVSA control components ..............................38

11. Validation ...................................................................................41
  11.1 Introduction ..........................................................................41
  11.2 Unit tests ............................................................................41
  11.3 Behave tests .........................................................................42
  11.4 Code review .........................................................................42

12. Deployment ................................................................................43
  12.1 Deployment of the KVIVSA component ...............................43
  12.2 ASD models test deployment strategy ................................43

13. Project Management .................................................................44
  13.1 Realization dimension ..........................................................44
  13.2 Project Planning and Scheduling .........................................45
     13.2.1. Backlog planning ....................................................45
     13.2.2. Gantt chart ............................................................45
  13.3 Realization strategy ............................................................46

14. Project Retrospective .................................................................47
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.1</td>
<td>Reflection</td>
<td>47</td>
</tr>
<tr>
<td>14.2</td>
<td>Design opportunities revisited</td>
<td>47</td>
</tr>
<tr>
<td>14.3</td>
<td>Process</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Glossary</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>About the Authors</td>
<td>53</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1. Overview of ASML TWINSCAN machine ........................................... 3
Figure 2. Overview of DCA pattern with emphasized project’s scope .............. 6
Figure 3. Data aspect from DCA pattern ....................................................... 7
Figure 4. Control aspect from DCA pattern ................................................... 8
Figure 5. ASD component [32] ...................................................................... 9
Figure 6. An SBS in the ASD:Suite [9] ......................................................... 10
Figure 7. Algorithm aspect from DCA pattern ............................................. 11
Figure 8. Stakeholders engagement flow .................................................... 13
Figure 9. Stakeholder analysis chart ............................................................ 14
Figure 10. Architectural Reasoning Model (ARM) ....................................... 25
Figure 11. Application dimension [4] ............................................................ 26
Figure 12. Construction dimension [4] ......................................................... 27
Figure 13. The generate stub dialog [7] ........................................................ 36
Figure 14. Verification results [6] ................................................................. 38
Figure 15. Sequence diagram for failed checks [6] ..................................... 39
Figure 16. Realization dimension [4] ............................................................ 44
Figure 17. Realization strategy ..................................................................... 46
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Stakeholder categorization</td>
<td>16</td>
</tr>
<tr>
<td>Table 2</td>
<td>Identified risks</td>
<td>19</td>
</tr>
<tr>
<td>Table 3</td>
<td>Reasons for following DCA pattern</td>
<td>28</td>
</tr>
<tr>
<td>Table 4</td>
<td>Reasons for using ASD-Suite</td>
<td>29</td>
</tr>
<tr>
<td>Table 5</td>
<td>ASD verification features [6]</td>
<td>37</td>
</tr>
</tbody>
</table>
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVSA</td>
<td>Improved Vertical Stage Alignment</td>
</tr>
<tr>
<td>DDS</td>
<td>Domain Data Services</td>
</tr>
<tr>
<td>DLS</td>
<td>Domain Logic Services</td>
</tr>
<tr>
<td>TIS</td>
<td>Tis plate</td>
</tr>
<tr>
<td>PARIS</td>
<td>Paris plate</td>
</tr>
<tr>
<td>LS</td>
<td>Level Sensor</td>
</tr>
<tr>
<td>SA</td>
<td>Stage Alignment</td>
</tr>
<tr>
<td>RA</td>
<td>Reticle Alignment</td>
</tr>
<tr>
<td>HSA</td>
<td>Horizontal Stage Alignment</td>
</tr>
<tr>
<td>VSA</td>
<td>Vertical Stage Alignment</td>
</tr>
<tr>
<td>MDE</td>
<td>Model-driven engineering</td>
</tr>
<tr>
<td>MDD</td>
<td>Model-driven development</td>
</tr>
<tr>
<td>SBS</td>
<td>Sequence-Based Specifications</td>
</tr>
<tr>
<td>MS</td>
<td>Measure Sequence</td>
</tr>
<tr>
<td>EDS</td>
<td>Element Design Specification</td>
</tr>
</tbody>
</table>
1. Introduction

This chapter gives a general introduction to the project by describing its context and the environment in which the project was performed. The chapter concludes with an overview of the structure of this report.

1.1 Context

The project “Scalable Model-Driven Software for Stage Alignment of ASML TWIN-SCAN Machines – Control and Domain Logic Services” was conducted by Anita Asprovska, as part of the Professional Doctorate in Engineering (PDEng) program. The program is offered by the Department of Mathematics and Computer Science of Eindhoven University of Technology in the context of 4TU.School for Technological Design, Stan Ackermans Institute.

The project was initiated by ASML, leader in the manufacture of lithography systems for the semiconductor industry. The project was conducted at ASML, as part of the Leveling Group (FC-062) within the Metrology Department in Eindhoven.

1.1.1. ASML metrology department

ASML manufactures machines for the production of integrated circuits. Because this company is quite large, it is divided in several departments, one of which is the Metrology department. Metrology comes from the Greek words “Metron” (measure) and “Logos” (study of), which means study of measurement. The expertise of this department is to exploit the measurements provided by multiple sensors from the machine, in order to improve its efficiency and accuracy. This project was conducted in this department.

1.2 Outline

This technical report describes the progress, process and results of this project. It starts with an extensive description of the context and problem analysis in the following Chapter 2. This chapter elaborates the project description and the project goal. Next, Chapter 3 defines the details of the domain of the project.

It continues with introducing the stakeholders involved in the project by presenting their concerns and responsibilities (Chapter 4). The following Chapter 5, defines the risk analysis and the risk management strategy.

After having the extensive analysis of the problem and the domain, the requirements that were agreed with the stakeholders are documented in the continuation of the report in Chapter 6. The system requirements are the basis for the next Chapter 7. This chapter elaborates on the customer needs and describes the application dimension by defining the most important use cases.

The subsequent chapter (Chapter 8) elaborates on the context of the system design. Here the construction dimension is described by explaining which and how some of the design decisions were made. Besides the rationale of the design decisions, this chapter summarizes in details the actual control component design. It explains the failure handling methods as well.
The report continues in elaborating about the verification (*Chapter 10*) and validation (*Chapter 11*) methods applied in the execution of the project. It also explains how the use cases and design were implemented and deployed, and which tools were used for that purpose (*Chapter 10 and 12*).

Finally the report summarizes and describes the process of managing the project (*Chapter 13*). It concludes with giving an overall retrospective of the project execution (*Chapter 14*). ■
2. Problem Analysis

Before diving into the project details, this chapter elaborates what is the problem that needs to be tackled by describing the project’s scope and goal. It moves forward to describing the future plans and summarizing the possible design opportunities.

2.1 Context

ASML is a provider of lithography systems for the semiconductor industry, manufacturing complex machines that are crucial to the production of integrated circuits or chips. The most complex and valuable product that ASML delivers is the TWINSCAN machine (Figure 1). The main application of this system is to transfer a pattern of an integrated circuit device from a photomask (reticle) onto a light sensitive chemical (resist) on the silicon wafer. The process can be compared to the process of taking a picture with camera. What is common in both processes is that the pattern in the resist is created by exposing the material in front of a light source, either directly or with a projected image using a photomask.

These kind of systems usually require nanometer accuracy at high speeds. The TWINSCAN system developed in such way that it achieves these high speeds and accuracy required by the modern semiconductor manufacturing. What is peculiar about this
generation of lithography systems is that they are able to simultaneously operate on two wafers. These two wafers are being processed in the system at the same time, one being exposed and the other one being measured. This way of working doubles the productivity but also increases the complexity of the entire system. To keep the accuracy of the measurement in such environment, measurements of a common reference are needed at both “measure” and “expose” sides of the system. Attaining this requirement by only using mechatronics is impossible. Besides all the sensors, there has to be mathematical and software modeling that helps realizing this objective. The leveling group within the metrology department is responsible for the modeling of the vertical measurement inside the machine.

2.2 Project description
This project was initiated as a first step of the process of redesigning the IVSA component that will eliminate the accumulated complexity in the existing implementation.

2.2.1. Project goal
The aim of this project is formulated and agreed with the stakeholders as follows:

The goal of this project is to address the challenge of redesigning the IVSA module by starting the development of the new software component which will follow the new DCA reference architecture using the model driven approach.

The current implementation of the component is complex. To prove this, static analysis of the code was done and based on the analysis the software department decided to start with building new IVSA component. The new implementation needs to address the most important design requirements scalability, maintainability, and robustness. These are the main challenges that led to this project initiation.

2.2.2. Project scope
In order to boost the development of the IVSA component, two PDEng trainees were hired to start the design and development of the software. One of the project was focused on the Control aspect of the DCA reference architecture and the Domain Logic Services. The second project (from this point on, referred to as Project 2) was focused on the Algorithm aspect of the architecture and the Domain Data Services. The Project 2 was also conducted by a PDEng trainee (Ianislav Mintchev). Both projects were conducted in parallel and the work was integrated continuously. The DCA pattern itself is elaborated in section 3.1 and Figure 2 from the same section, depicts the scope of this project referring the DCA pattern visualized by the gray area.

Based on the time constraints and the development resources, only one scenario from the IVSA function was chosen for the scope of this project. This scenario is the basic use case of the IVSA.

2.3 Roadmaps

2.3.1. Business Roadmap
ASML is the key player in the industry of integrated circuit production. The company is striving to enable its customers to increase the value and reduce the cost of their chips. By doing this, the semiconductor industry carries out the fast-paced development of the integrated circuits and in this way supports the progress in a lot of technologically related fields. The main goal of the business is to keep Moore’s law alive for many years to come, and hence boost the mass production of the chips every year.

2.3.2. Technology Roadmap
Keeping up with the business goals requires constant improvement of the system under development in terms of efficiency, scalability, robustness, performance and many
more aspects. For that reason the ASML departments are constantly working on finding new technologies and methodologies that would help them achieving these goals. The new ASML metrology software approach uses the new Data, Control and Algorithm (DCA) reference architecture. This architectural pattern is based on the separation of data, control and algorithms. By using such separation of concerns, the responsibilities are made clear and allow different tools and modelling techniques to be used for each layer. Alongside this approach, they are also trying to shift the development towards the use of model-driven engineering. The principal goal for the technology roadmap is to develop a modular and consistent system that would integrate various models into one coherent entity.

2.3.3. Product Roadmap

After the analysis conducted over the software module for Vertical Stage Alignment (VSA), the result showed that the complexity of the component is descoping any functional extension. In order to stop this, the software group in the metrology department set a goal to replace the old VSA component after the new component is fully developed. In the interest of the new component evolution, the development needs to go slowly, step by step, each step ensuring and validating the product. To boost the development in the beginning, two OOTI students were hired to start with the development of the main scenario and build the basic ground for the development of the module. The delivery should provide a new data model that is supported by the control and algorithm aspects of the system. Moreover, the idea is to employ the new reference architecture using the model driven techniques.

2.4 Design opportunities

Addressing the problem, analyzed in this section, three design opportunities were identified, namely: methodical approach, functionality and complexity. The identified design opportunities are revisited in section 14.2.

2.4.1. Methodical approach

First design opportunity that is identified is the methodical approach. This opportunity was chosen because proven technologies and methods needed to be followed in the design process of the project. Additionally, one very important goal is that the requirements are proved to be satisfied as defined by the stakeholders. The following indicators for choosing this design opportunities were recognized:

- Component modeling
- Separation of concerns
- Model-driven engineering approach

2.4.2. Functionality

Second design opportunity chosen was functionality, since the main functions that need to be fulfilled by this artefact are clearly defined. Indicators for choosing this opportunity were:

- Satisfying the requirements for the lot production use case
- Satisfying the stakeholders concerns (responsible for the maintenance of the deliverables)
- Reusability

2.4.3. Complexity

Third design opportunity identified is the complexity. While doing the problem analysis it was concluded that we have to deal with complex problem that requires a complex domain knowledge from various disciplines (for example physics).
3. Domain Analysis

Following the problem analysis is the chapter that extends this analysis by defining the domain of the problem. It gives and introduction to the reference architecture.

3.1 Reference Architecture

Separation of concern is a design principle that enforces building modular software by dividing it in separate sections with specified concerns of interest. Concern in the software technology is a term that represents a set of information that affects the computer program. This principle is achieved by encapsulation, which is a means of information hiding. The advantage of employing this principle is enormous in terms of effective, fast development and maintenance, reusability of the modular sections and modification only on a specific module [3].

When looking into software development, there are at least three things that an engineer needs to tackle:

- Data persistence
- Decision logic
- Data transformation

Trying to solve all of these things in one go can become really complex. In order to decrease the complexity the best way to approach such development is to separate these three aspects and tackle each one of them one by one. Since the development of the TWINSCAN software is quite complex, the Software Architecture Group has detected the following three aspects:

- Data
- Control
- Algorithm (durative actions)

This separation of concerns led to the architectural pattern called DCA (Data Control Algorithm) [14][15], depicted on Figure 2.

![Figure 2. Overview of DCA pattern with emphasized project's scope](image)
3.1.1. Data

In every software product the data plays an important part since most of the systems basically perform operations over some data. The data aspect encloses the production, update, access and destruction of the data needed by the system. In order to avoid exhausting the system resources and most importantly having organized and well-managed data, it needs to be carefully separated, designed and stored in the memory. To have data that is easy to manage is really important because it affects the maintainability of the system. If the data is not well organized, we might face the problem of spending too much time on constantly aligning data variables which hold the same data in different places in the system.

When using the DCA pattern, the data aspect needs to be separated from the rest. One way of achieving this is by using the Model Driven Engineering (MDE) approach. By using this approach it is easy to capture the domain knowledge in formal, high-level model. In this way, a data model can be created using the data tool called ASOME suite. The language used for specifying the data model using this tool is based on the concepts of domain driven design, which includes the concepts such as Entity, Value Object, Association, Composition and Multiplicity. The language uses similar concepts with the UML class diagrams as well. The ASOME tool provides code generation from the data model, besides designing the domain model.

Important distinction for the entity concepts is the entities whose creation is triggered by the algorithms, and the entities whose creation is triggered by the control components. Since the control aspect is data unaware, the entities which are created by the control represent an action that needs to be performed as decided by the control component. Additionally, the algorithmic entities add details that do not need to be known by the control. This distinction is essential for achieving separation between the control and the data. Separating the data from the control avoids a state space explosion in the control domain, within the context of formal verification to the control components.

The data layer, Figure 3, contains the Domain Data Services (DDS) and the Domain Logic Services (DLS) aside from the domain model. The DDS provide interface for creation, access, update and deletion of the data. This interface is used by both Algorithm and Control aspects. The algorithm layer is able to create, update and access the data, whereas the control is responsible for the lifetime of the data, meaning creation and destruction. As for the DLS, they provide interface for evaluation questions. These services extract decision information from the domain model. Hence, the client of these services is the control aspect, which is responsible for taking the decisions. Since the control is the focus of this project and it uses the DLS interface, it also comes in scope of this project. The DDS interface is also part of the scope of this project because it is used by the control to create and destroy so called control entities.
3.1.2. Control

Another aspect that can be identified in almost every system is the control aspect, Figure 4. The control logic takes care of scheduling and planning the execution of the tasks in the system. Since one task can have multiple subtasks, and in order to optimize the utilization of the system’s resources, the execution of these tasks needs to be planned carefully. This planning becomes complex really fast. Verification of such a system also becomes computationally intensive using the formal, mathematical means to verify the specified requirements. In order to prevent this kind of complexity, separating the aspects and verifying these aspects in isolation is advised. For example, when verifying control aspects, state space analysis is used. This means that the control behavior of the system is described by the means of state machines. When the guards on the transitions between states are based on data, the state space explodes and becomes very large, which means that the analysis can take a long time or it can be even infinite. This is a rationale for keeping the control unaware of the data.

For the control aspect it is desired that it is responsive (not strictly needed at all time). A module that is responsive does not block its clients; in contrast, it reacts quickly and positively. The control can get events at any time and it should respond to those events, hence it cannot wait for any actions that take time (for example algorithms). Therefore those actions are also separated from the control logic.

The control layer needs to be deterministic, reacting on each event trigger or call. It can be described as a sequence of high level actions by using the means of flow diagrams, sequence diagrams, decision diagrams, data flows or state machines.

The control aspect from the DCA pattern in ASML is developed with the tool called ASD:Suite, which is developed by the company Verum [5][6][7]. The focus of this project is based on the control aspect and its integration with the algorithm and data aspects.

ASD:Suite

ASD stands for Analytical Software Design, which is a technology that applies the model-driven design with formal methods to create mathematically verified software designs. The ASD:Suite is the software engineering platform that embodies this technology. That is to say, the platform facilitate the mathematical verification of the software designs before any code is written. When the designs are verified, the platform enables generation of a code out of the verified components. The ASD platform is suitable for designing reactive systems (control behavior) by modeling the states of the system, all the events that can happen and the resulting action after each event.

ASD is a component and model-based development environment. The smallest unit of decomposition in ASD is the ASD component, which is described by models that describe its external and internal behavior. Each ASD component is specified using an
interface and a design model. The interface model (.im file extension) is implementation-free specification that defines the externally visible (black-box) behavior of an ASD component. This model has the following characteristics:

- Defines what the component does, not how the component will do it.
- Shows externally visible behavior and hides the implementation details.
- It represents an abstraction of a component that every compliant component is required to implement.
- It is defined in terms of events that are exchanged between the component and its clients. This is described by method signatures and externally visible behavior specified in Sequence Based Specification (SBS).
- It is used to generate code for the interface declarations.

As for the design model (.dm file extension), it defines the implementation of the external visible behavior as defined by the interface model and the internal behavior using the interface models of the used components. This model has the following characteristics:

- Describes how the component implements its behavior.
- Design models are always deterministic.
- It is defined as events that are exchanged between the component and its clients and servers.
- It is used to generate code for the executable logic.

An ASD component implements or provides a service that is used by its clients. On the other side, an ASD component uses or requires services that are implemented by its servers. This is displayed in Figure 5. [32]

![Figure 5. ASD component][32]

The Component blue rectangle represents the design model. The Provided rounded rectangle represents the provided interface model of the component. The Required1 and Required2 rounded rectangles represent the required interface models of the component. [32] “This implemented service (Provided) is exposed by means of application interfaces through which clients can send call events. The ASD component can respond to a call event on an application interface by means of a reply event on the same application interface and notification events on notification interfaces. In this process, an ASD component can also invoke used services (Required1 and Required2) that are implemented by other components: the servers of the ASD component. Collectively, the services between a component and its clients and servers form an imaginary border, called the component boundary. Information crosses the component boundary in the form of events." [7]

“A component "knows" only information passed into it across the component boundary in the form of the triggers it receives. A trigger can be:

- A call event from a client through an application interface;
- A reply event from a server through an application interface;
- A notification event from a server through a notification interface.
Similarly, a component exposes information to its clients and servers across the component boundary in the form of the actions it sends. An action can be:

- A call event to a server through an application interface;
- A reply event to a client through an application interface;
- A notification event to a client through a notification interface.

An interface model is defined in terms of only those events that pass between a component and its Clients. A design model is defined in terms of events that pass between the component, its Clients and its Servers. Within ASD, both interface models and design models are defined in the form of Sequence-Based Specifications (SBS). Figure 6. Behavior is specified in a tabular form as a total Black Box function, by mapping all possible sequences of triggers to the corresponding actions.” [7]

<table>
<thead>
<tr>
<th>Interface</th>
<th>Event</th>
<th>Guard</th>
<th>Actions</th>
<th>State Variable Updates</th>
<th>Target State</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotActivated (initial state)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 AlarmSystem</td>
<td>SwitchOn+</td>
<td></td>
<td>WindowSensorSensorActivate; AlarmSystem.On</td>
<td></td>
<td>Activated_On</td>
</tr>
<tr>
<td>5 AlarmSystem</td>
<td>SwitchOff</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 WindowSensorSensor_ND</td>
<td>DetectedMovement</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 WindowSensorSensor_ND</td>
<td>Deactivated</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 TimerTimerCB</td>
<td>Timeout</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activated_On</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 AlarmSystem</td>
<td>SwitchOn+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 AlarmSystem</td>
<td>SwitchOff</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 WindowSensorSensor_ND</td>
<td>DetectedMovement</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 WindowSensorSensor_ND</td>
<td>Deactivated</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 TimerTimerCB</td>
<td>Timeout</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. An SBS in the ASD:Suite [9]

3.1.3. Algorithm

Like stated previously, the systems usually manipulate data by consuming and producing data. The data that has been consumed is then transformed in the needs of the system. The complexity of such transformation can vary from a simple linear mathematical operation to solving an optimization problem. Because these transactions usually take time, they are called durative services.

The algorithmic layer (Figure 7) includes the durative actions that are performed in a system in order to transform data that is either consumed or already existed in the system. This includes the algorithms depicted with the mathematical function on the Figure 7, and the external services depicted by the gears. The algorithms are basically an atomic action that has input and output data, but it also defines how to reach from the input to the output data. Since the algorithms can be quite complex they are also becoming complex to test. It is impossible to test every input value of the algorithms. Because of this they need to behave as pure mathematical functions. In order to test such algorithms, it is recommended that they are tested with developed test vector which can be tested against the desired outcome.
What is specific about this layer is that the algorithms need to be stateless in order to be able to test them in isolation. On the other hand, the external services are usually actions performed on physical resources, hence they can be stateful.
4. Stakeholder Analysis

In this chapter, the stakeholders that are affected by a decision or the outcome of this project are identified and an engagement strategy is defined per stakeholder.

4.1 Introduction

The stakeholders for this project come either from ASML or the Eindhoven University of Technology. In the process of identifying the stakeholders, they are divided in two groups: direct and indirect stakeholders. The first group of stakeholders includes the stakeholders that have high influence and high interest in the project. The direct stakeholders have direct impact on the project and these stakeholders are the ones that need frequent update and involvement in the process. The indirect supervisors are important for the project because they facilitate the execution indirectly, by providing domain knowledge or providing the necessary means for execution.

According the organizational structure of the company where the project was conducted, the stakeholders were engaged according the flow described in Figure 8. This flow shows how the stakeholders were addressed in order. First contact person in the chain is the Primary Supervisor followed by the Secondary Supervisor. In case further engagement is needed the Project Owner was included. As the ultimate point of engagement was the Project Coordinator.

All stakeholders identified for this project are presented in Figure 9. This figure shows the influence that a particular stakeholder has on the project in relation with his interest in the outcome of the project. The strategies for managing the stakeholders are derived from this chart. If the stakeholder has high interest and high influence, then the stakeholder is engaged frequently and is involved in the decision making. If the interest is low but the influence is high, then in that case the stakeholder is engaged only on topics relevant for him and an effort is made to increase his/her interest. If the interest is high and the influence is low, then the stakeholder is engaged when his interest areas are under consideration. For the stakeholders who have low interest and low influence the strategy was to keep them informed only when certain activity has impact on their work or when their knowledge is needed.
4.2 Stakeholders

I. Javad Vazifehdan – Project Coordinator
Javad is the project coordinator. He is an indirect stakeholder for the project. He works as a project manager in the ASML leveling group and his main role in the project is to provide administrative support as needed. His concerns involve:

- ensuring that the deadline is met
- receiving feedback on the current way of working in the company
- receiving recommendation for the aspects that need improvement

II. Lewis Binns – Product Owner
The product owner is the initiator of the idea behind the project; hence he is one of the most important stakeholders with highest interest and huge influence on the project. He is the functional cluster architect of the leveling group (FC-062) in the metrology department.

Since it was his idea, this stakeholder is involved regularly in the project and in decision making. His concerns include:

- Advance on the product roadmap
- Improve the design of the component
- Increase the software quality by employing MDE way of working and DCA pattern
- Successful integrated (with Project 2) presentation of one use-case to the employees

III. Santiago Cifuentes Costa – Primary Supervisor
Santiago is part of the software group in the metrology department where he works full time on a project in the same domain as this project. He is a promoter of the DCA pattern and an expert in implementing the control logic. The primary supervisor is responsible for monitoring the progress of the project. He supports the trainee in the execution of the project by:

- giving feedback on the trainee’s work
- providing guidance and domain knowledge
- assisting in communication with indirect stakeholders

Apart from his responsibilities, the primary supervisor is interested in:

- the quality of the deliverables
- the advantages of separating concerns (DCA)
- the trainee’s progress
IV. Edwin Jansen – Secondary Supervisor
The secondary supervisor is one of the directly involved stakeholders in this project. He is part of the software group in the department and works full time on one of the ongoing projects. His primary role is actually being a primary supervisor of another OOTI trainee working on the Algorithms aspect in the development of the IVSA. As secondary supervisor on this project he is concerned mostly about alignment of these two projects. His role is to make sure that the two trainees do not block each other’s work and our progress is steered towards a common goal. Apart from these responsibilities Edwin is facilitating this project by providing domain knowledge. Based on his experience he provides guidance in the data modeling aspect of the project and he is involved in the integration process.

V. Arjan Holscher – Group Leader
Arjan is the group leader of the leveling group (FC-062) in the metrology department where this project was being conducted. Even though his interest in this project is not very high, he has strong influence on the execution because of his position in the company. His main concern about this project is the value that this project will bring to the way of working in his group. He is not interested in the details of the execution but the outcome and the impact that this project will bring to the group. Another concern of his is the resource allocation. This means that the group leader decides which person is going to supervise the trainees.

VI. Ianislav Mintchev – PDEng Trainee (Algorithms & DDS)
Another important stakeholder of the project is the PDEng trainee who is working on the Project 2 focused on the supplementary segment of the DCA pattern (Algorithms and Domain Data Services). His concerns for the project involve:
- On time integration in order to deliver his part of the project
- High level understanding of the project
- Change impact on his work
- Involvement in the decision making that affect his project

VII. Anton Wijs – TU/e Supervisor
Independently from the company supervisors, the Technical University of Eindhoven (TU/e) as a stakeholder is represented by Anton Wijs. He is an Assistant Professor of Parallel Software Development at the Model-Driven Software Engineering group at TU/e. The university supervisor assists the trainee during the execution of the project by:
- Monitoring the progress and the quality of the work
- Facilitating the trainee’s model driven design decisions
- Giving feedback to the trainee
- Reviewing the technical report

VIII. Yanja Dajsuren – TU/e Program Director
The Program Director is one of the most important stakeholders for this project. As a Program Director, Yanja has high interest and influence in each of the trainees’ projects. She is responsible for acquiring relevant project proposals from the companies in the software industry; hence her concerns go in two directions. On one hand, she expects the trainee to gain meaningful experience working in the industrial environment. On the other hand, her concern is that the company is satisfied by the execution of the project and their requirements are met at the end, so the company and the university can maintain their collaboration.

IX. Lianne Muijers – Functional Expert
Lianne is the author of the EDS document describing the detailed functionality of the IVSA component. She worked within the functional group in the metrology department. Her main role in this project is providing the source of the functional requirements towards the execution of the project. She is an indirect stakeholder because she is currently working in different a department in the company which means her interest in the project is really low.
X. Hugo Looijestijn – Functional Expert
Hugo is Lianne Muijers successor, providing the functional support needed for the execution of this project. He works in the leveling group as a functional expert. His interest in this project is neutral, because he is only contributing to the project by clarifying parts of the EDS document, which are not described fully. Therefore, Hugo is part of the indirect group of stakeholders which facilitate the project with providing domain knowledge.

XI. Anita Asprovska – PDEng trainee
Anita is the PDEng trainee that is responsible for the entire execution of the project including:
- Planning the project
- Managing stakeholder’s expectations
- Defining the project scope
- Analyzing, designing and implementing the system requirements
- Verifying and validating the result
- Writing the final report
Her concern is successful execution of the project, which means on time delivery of the agreed product so that she can obtain her degree.

Table 1. Stakeholder categorization

| Stakeholder            | Role                  | Group     | Engagement method          | Engagement frequency
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Javad Vazifehdan</td>
<td>Project Coordinator</td>
<td>Indirect</td>
<td>Demo meeting</td>
<td>Regular, Bi-weekly</td>
</tr>
<tr>
<td>Lewis Bins</td>
<td>Project Owner</td>
<td>Indirect</td>
<td>Demo meeting</td>
<td>Regular, Bi-weekly</td>
</tr>
<tr>
<td>Santiago Cifuentes Costa</td>
<td>Primary Supervisor</td>
<td>Direct</td>
<td>Stand-up meetings PSG meetings Pull request reviews</td>
<td>Regular, Daily</td>
</tr>
<tr>
<td>Edwin Jansen</td>
<td>Secondary Supervisor</td>
<td>Direct</td>
<td>Stand-up meetings Pull request reviews</td>
<td>Regular, Daily</td>
</tr>
<tr>
<td>Arjan Holscher</td>
<td>Group Leader</td>
<td>Indirect</td>
<td>e-mail</td>
<td>On demand only</td>
</tr>
<tr>
<td>Ianislav Mintchev</td>
<td>PDEng Trainee</td>
<td>Direct</td>
<td>Stand-up meetings</td>
<td>Regular, Daily</td>
</tr>
<tr>
<td>Anton Wijs</td>
<td>TU/e supervisor</td>
<td>Direct</td>
<td>PSG meetings Face-to-face meeting</td>
<td>Regular, Bi - weekly</td>
</tr>
<tr>
<td>Yanja Dajsuren</td>
<td>PDEng Program Director</td>
<td>Indirect</td>
<td>No regular contact</td>
<td>On demand only</td>
</tr>
<tr>
<td>Lianne Muijers</td>
<td>Functional EDS Author</td>
<td>Indirect</td>
<td>No regular contact</td>
<td>On demand only</td>
</tr>
<tr>
<td>Hugo Looijestijn</td>
<td>Functional Expert</td>
<td>Indirect</td>
<td>Face-to-face meetings</td>
<td>Monthly meetings (the beginning of the project)</td>
</tr>
</tbody>
</table>

Table 1 describes the categorization of the project stakeholders along with the engagement strategies for each of them. Stakeholders who are actively involved in the execution of the project, are part of the direct stakeholders group. This group of stakeholders frequently receives updates on the progress and the design decisions. The engagement of this group is either through a PSG meeting or a stand-up meeting. Depending on the
frequency, each stakeholder is engaged in different meetings. Stakeholders who have high influence on the project but are not actively involved in its execution are part of the indirect group of stakeholders. The engagement of this group of people is less frequent than the directly involved stakeholders because their interests do not involve details of the execution of the project.
5. Feasibility Analysis

After analyzing the problem and the domain, a feasibility analysis was performed in order to identify the risks that may occur during the lifetime of this project. The result of this analysis is revealed in this chapter.

5.1 Risks

During the lifetime of a project there are number of uncertainties that can occur and impact the execution of the project in a negative way which is unpredictable. Even though it is impossible to predict all the uncertain events that can affect a project, it is really important to prepare and minimize the occurrence or the impact of some events which can influence the project. In order to deal with these uncertainties, the risk analysis practice was used in the management of this project.

The risk analysis and risk management were conducted throughout the lifespan of the project. These practices involved several recurring steps:

- risk identification
- risk evaluation
- risk mitigation
- risk monitoring

After the domain and problem analysis the process of identifying the risks began. The reason why this task was one of the tasks early in the stage of the project is that the risks need to be identified as early as possible, so there was time to act accordingly if they occur. The next stage was the evaluation of the risks in terms of probability of occurrence and the risk impact. With regards to probability of occurrence the risks are evaluated in four categories:

- High
- Medium-high
- Medium-low
- Low

As for the risk impact the risks are categorized in three groups:

- High impact
- Medium impact
- Low impact

Afterwards, the mitigation strategy was defined for each identified risk. Finally, the process of monitoring and revisiting the risks was conducted. All these steps were repeated multiple times during the project.

The results of these phases is summarized in Table 2.

Table 2. Identified risks

<table>
<thead>
<tr>
<th>ID</th>
<th>R.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Integration with the algorithms is not possible</td>
</tr>
<tr>
<td>Probability of occurrence</td>
<td>Medium-low</td>
</tr>
<tr>
<td>ID</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>R.02</td>
<td>Late changes to the goal, scope and requirements of the project.</td>
</tr>
<tr>
<td>R.03</td>
<td>Documentation unavailability</td>
</tr>
<tr>
<td>R.04</td>
<td>Supervisors unavailability</td>
</tr>
<tr>
<td>R.06</td>
<td>Development tools unavailability.</td>
</tr>
<tr>
<td>R.06</td>
<td>Misunderstanding the requirements</td>
</tr>
</tbody>
</table>

Originally these were the risks that were defined and documented during this project. Aside of these risks there were some issues that occurred which were not planned such as:

- Difficulties in scheduling meetings on time
• Delaying task execution by facing implementation issues
• Development environment issues

For the risks that were faced in this project, the impact was not as severe as anticipated. The mitigation strategy defined for each of the risks was followed and the impact was reduced than originally estimated. Even the issues that occurred, which were not expected to happen, did not have severe impact over the successful execution of this project.
This chapter summarizes how the requirements elicitation process was conducted.

6. System Requirements

6.1 Introduction

One of the first phases in the beginning of a project is the process of requirements elicitation which is part of the requirements engineering. Requirements engineering is defined as: “The subset of systems engineering concerned with discovering, developing, tracing, analyzing, qualifying, communicating and managing requirements that define the system at successive levels of abstraction”[1]. Before designing a system there has to be a knowledge of what the future needs of that system might be. The process of requirements engineering is concerned with finding information about such needs and the associated changes. It is concerned with identifying what should be designed in order to meet some perceived future need. [2] This doesn’t mean that this process is done only before starting the system design. In most of the dynamic projects, like this one, the requirements can come or change in any time in the lifetime of the project. One of the reasons is that in the beginning of the project it is not easy to estimate accurately what and how much can be delivered in a limited amount of time. Another reason is that not all the needs are clear in the beginning but are rather ambiguous. In this project the system requirements were refined and revisited multiple times during the execution.

Prior to the phase of requirements elicitation, the stakeholder analysis was conducted. This analysis was essential for the requirements gathering process, since the stakeholders are one of the most important sources of information. The system requirements for this component are gathered using multiple elicitation techniques that involved the identified stakeholders for example:

- brainstorming meetings
- individual interviews
- focus group

Apart from using these techniques, other methods were also used to elicit requirements. These methods did not include the stakeholders directly, but they were focused on exploring the existing resources. Such techniques that were used were:

- interface analysis
- document analysis
- use cases

The output of this phase is not part of the public report.
7. System Architecture

In this chapter, the architectural reasoning process is described. It focuses on the application dimension by depicting the use cases which define the main function that needs to be provided by the system.

7.1 Introduction

An architecture of a system is produced in order to build a consistent and maintainable system that complies with the stakeholders concerns and the design constraints. The Architectural Reasoning Model (ARM) facilitates the decision making process and the process of analyzing the taken decisions. The purpose of the ARM is to assist in the process of taking decisions with regards of the application, construction and the realization aspect of the system. Another application of this model is to make sure that there are no contradicting statements for any of these system aspects. This model is also used to analyze the changes in customer needs, technology, development process or development organization. This model is used in creating the architecture of the IVSA so that the decisions taken are consistent with the customer needs. [4]

Figure 10 shows the application of ARM to a software system. The outer circle of the figure represents the environment of the system, while the middle circle represents the boundary between the system and its environment. The circle in the center of the figure represents the system itself.

The ARM describes three dimensions of reasoning about the architecture of a system, application, construction and realization dimension. Therefore, the architecture of this system is also described in the same manner. The application aspect is analyzed in the
following chapter, as for the construction and realization dimension, they are described in the System Design, Implementation and Project Management chapters.

7.2 Application dimension
The application dimension from the ARM represents what the system has to offer to the users or how it impacts them, depicted on Figure 11. [4] This dimension addresses the customer needs and the use cases which IVSA needs to satisfy. The customer needs are translated into functional and the application of those customer needs are described as use cases.

In order to satisfy the goals of the project and the customer needs, it is decided that the main purpose of this project will be the design and implementation of the main IVSA use case. The remaining use cases of IVSA are out of the scope for this project because of the time and resource constraints.

The detailed description of the use cases is not part of the public report.
8. System Design

Following the application dimension analysis, comes the reasoning for the system construction. This part elaborates the rationales for the design decisions as well as a detailed description of the system construction regarding the control aspect. The chapter concludes with explanation of the failure handling process.

8.1 Construction dimension

With modeling the problem, the domain, and the customer needs, basis for designing the component is established. Along the construction dimension it is described how the system works and how it uses the existing constructions or other systems. Here, the connection between what the system needs to provide and how it satisfies those needs is described. Figure 12 depicts what the construction dimension represents. [4]

In the construction dimension for this system we reason about:
- the design of the component
- the interfaces between the component and its clients (in the environment)
- the interfaces between the structural elements in the component’s design
- the responsibilities of each structural element
- the design patterns and principles that the elements need to adhere to
- the failure handling
- the way of addressing the non-functional requirements
- the technology choices

The reasoning for all of these aspects is elaborated in the following sections.
8.2 Design decisions

Part of construction dimension is to relate the customer needs with the existing technologies and describe how those technologies will solve the problem. In this section, the design decisions that were taken in the beginning of the project are described.

8.2.1. Rationale for choosing DCA pattern

One of the first decisions taken was to follow the DCA architectural pattern. As mentioned in the domain analysis section 3, every software system can be decomposed into three features Data, Control and Algorithms. The DCA architectural pattern that facilitates this decomposition can be applied to every system which modifies data using durative actions in certain context. The reasons why this pattern is being used in the design of this system are shown in Table 3.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addressing requirements</td>
<td>Using this pattern addresses several system requirements. One such requirement is to build maintainable system. Separating the different aspects of the system helps in maintaining each of the parts separately. This means that the engineers would not need to modify the algorithms and data if something in the decision logic happens, or vice versa. Another requirement that is addressed with this pattern is the scalability. The separation of concerns in the system leads to components which have single purpose and those components are easily reused in different context within the system. In other words, the control logic should always be responsive, which means that it can accommodate the growth from the incoming requests.</td>
</tr>
<tr>
<td>Existing components follow DCA</td>
<td>Currently the TWINSCAN software is build out of many components, each responsible for certain functionality. Some of those components are already following the DCA pattern. In order to adhere to one of the goals of the software department, which is to have one coherent and consistent model of the system, it was decided that this component will follow DCA pattern as well.</td>
</tr>
<tr>
<td>Developers are familiar with DCA</td>
<td>The purpose of this deliverable is to represent the basis of the IVSA component which will be extended in future to support the rest of the IVSA use cases. Because of this it is important to take into consideration, which technologies are already known to the developers that need to later extend this module. Since the software group in the metrology department already has experience with building systems that adhere the DCA pattern it was decided that this component will use the same technology.</td>
</tr>
<tr>
<td>Decreased verification time</td>
<td>Separating the control data and algorithms helps a lot in decreasing the complexity of the code and make the process of testing this kind of systems easier. When testing together the control data and algorithm it is complicated to detect where the problem is, since all of these aspects can be quite complex by itself. On the other hand, when testing in isolation for example the control, the state space stays as small as possible and because of that the verification process does not take a lot of time. Additionally, when verifying the domain data it is easy to verify with the functional experts without the need of knowledge of the decision logic. Finally, when verifying the algorithms, the duration of the process can decrease since the algorithms are only verified against the</td>
</tr>
</tbody>
</table>
input and outputs that they consume and produce. Which brings us to the conclusion that the overall time to test the system is less compared with testing all of these aspects together.

8.2.2. Rationale for choosing ASD

An important technology decision that was taken also in the beginning of the project was to use the ASD:Suite for developing the models of the control behavior. Table 4 explains the rationale behind choosing ASD tool for design and development of the IVSA component.

Table 4. Reasons for using ASD:Suite

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASD vs Dezyne</td>
<td>Dezyne is the new version of the ASD:Suite. This new version includes simulation tool which is not the case with ASD. Being newer version than its predecessor offers better features. One issue with this tool is that it is not approved for production by ASML. In order to approve new tool by the company, requires a lot of time investigating, following corporate standards, which will take much more time than the whole period allocated for this project. On the other side, the ASD:Suite is already production approved by ASML, which means it can be used for projects that are meant for production. Because the deliverable from this project is going to be used in production (after being extended), it was decided to use the approved tool. Since this is not the case with Dezyne, this proposal was refused.</td>
</tr>
<tr>
<td>Component based development</td>
<td>“Component-based development (CBD) is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems.” [17][18] For ASD tool this translates into designing software systems that consist of ASD and foreign components. “An ASD component represents a common unit of architectural decomposition, specification, design, mathematical verification, code generation and runtime execution”. [7] The foreign components are the ones that are being developed without using ASD, for example handwritten code, hardware, third party components. Using the interface and design model to specify the ASD component, it enables building component based systems with well-defined interfaces. Additionally, this practice of developing component based systems has advantages regarding the maintainability, reusability and the distribution of development between different developers. In particular, ASD component based development is suitable for designing loosely coupled components with a clear boundary which contributes to ease of the process of finding problems and modifying the system modules.</td>
</tr>
</tbody>
</table>
| Model-driven development | Analytical Software Design, the technology used by ASD tool, is the application of the model-driven design with formal methods in which the software systems are constructed from mathematically verified software components. The tool uses the ASD modeling language which allows creating two types of models, interface and design models. [5] Choosing this tool contributes to the common goal of the software team
which is trying to shift to MDE in order to build coherent system and using this approach try to decrease the time for verification and validation of the system.

<table>
<thead>
<tr>
<th>Verification tool integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One of the advantages of the ASD is that it has a verification tool integrated in the suite.</strong> “The ASD:Suite is used to detect errors in interface models and design models. Removing these errors ensures that the generated code works and that the resulting system has certain desirable properties, such as deadlock-freedom. The verification focuses on verifying properties that are hard for humans to verify. These properties mainly concern ordering of events, asynchronous behavior, deadlock and livelock.” [6] This process of verification helps verifying the models even before any code has been written. After the verification, code is generated out of the models, which guarantees that the generated code is for example, deadlock free. Verifying the models guarantees that the system is consistent, complete and robust.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Consistent deliverables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One of the most important aspects of ASD technology is the semantic equivalence between what is specified in the ASD models, the formal representation of those models used in the formal verification of the models and the runtime behavior of the generated source code.</strong> [5] The process of building a component using ASD is the following. First, the requirements are translated into well-defined interface models that describe what the component provides. Next, the design models are modeled in order to describe the behavior of the component, that adheres to the specified interface model. After finishing the behavior modeling, the ASD models are formally verified in order to eliminate errors that are hard to detect by a human. When the models are verified and checked for correctness and completeness, the code is being generated out of those models. Since the models are already verified, it can be concluded that the generated code is also correct and complete. This way of working ensures consistencies between the design and the implementation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reduced implementation and verification time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>With the current development methods, problems can arise easily since there are no formal means of verifying the design and architecture. This leads to problems later on in the development process, because the errors in the design are propagated in the implementation which is only discovered in the process of validation. After the errors are detected, the process is repeated until the right system has been build. This takes a lot of time. While, using ASD to design the behavior and verify formally those models, reduces the time for validation. Since the design has been formally checked, that leaves us with less error prone system to develop. Additionally, because of the code generation functionality the implementation time is also decreased. Only the hand written glue code that integrates the algorithms and the data with the control needs to be added.</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Addressing requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Using ASD to model the behavior, addresses the most important aspects for this project (scalability, maintainability and robustness). Because the current implementation of IVSA is facing the problem of maintainability and extendibility, modeling the new design of IVSA using ASD aims to improve those aspects. The ASD models are easy to maintain, since it is easy to find which model is responsible for which part. The models are also scalable which means they can be easily reused.</strong></td>
</tr>
</tbody>
</table>
Finally, because of the formal verification that checks the system in order to eliminate the problems which can be difficult to detect.

8.2.3. Following SOLID principles

SOLID are one of the most commonly used design principles in software development. It is an acronym which stands for the following design principles: [8]

- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion

In the rest of the section it is described which of these principles are being followed and how it is implemented in the design of the IVSA control logic.

**Single Responsibility Principle (SRP)**

This design principle states that every module or class should have responsibility over a single part of the functionality and that responsibility should be entirely encapsulated by the class. In order to avoid creating a bulky class with a lot of responsibilities, this principle should be taken into account. If we have such class that takes more than one responsibility, it means that every single change will impact that class, and later on impact the testing of the same class as well. If the SRP is followed, the classes will become compact and neat and all changes will only impact the responsible class. Note, that this principle stands for software components as well. [9][10] The principle is somewhat similar to the Separation Of Concern design principle, which is also being followed in this design (application described in section 8.2.1.). The SRP is followed when modeling the ASD interface and design components in such way that each component is responsible for one aspect of the behavior logic.

**Open/Closed Principle**

This principle states that a software entity should be open for extension but closed for modification. It can be applied for entities such as class, module or function. Being open for extension means that once a change is required, the entity is extended in order to add the new functionality rather than modifying it. This is useful because if the entity is changed that means it requires testing of the whole system and the entity’s clients rather than just testing the extension. [9][11] The open/closed principle is followed when designing the control in such way that the component is being designed using ASD models so that in future it can be extended with the other use cases of IVSA besides the main use case.

**Liskov Substitution Principle**

This is a principle in the object-oriented programing stating that, if B is subtype of A, then objects of type A can be replaced by objects of type B without altering the desired properties of the program. [33] Because this project was not focused on the definition of data/object types, this principle was not taken into consideration.

**Interface Segregation Principle (ISP)**

This principle states that no client should be forced to depend on methods it does not use. “ISP is intended to keep a system decoupled and thus easier to refactor, change or redeploy”. [9][12] ISP is being followed when designing the ASD components, in order to decouple responsibilities in the components it is decided to use different ASD models different segments.

**Dependency Inversion**

This principle states that high-level modules should not depend on low-level modules, instead both of the modules should depend on abstraction. Moreover the principle
states that the abstraction should not be dependent on details, but the details should depend on abstraction. [9][13] The principle is followed by defining abstraction between the control logic and the algorithms implemented in the leaf nodes of the ASD models.

8.3 Control design

The control aspect from the DCA architectural pattern is used to capture the behavior logic of the system, define its interfaces and describe the realization of those interfaces. It is used to describe the decision logic of the form ‘If event X occurs, then action Y is performed’. The control component receives requests that are often related to a certain context, which is defined by an entity. We refer to those entities as Control Entities. How those entities are been created is described in section 8.3.1. Like mentioned previously, since the control should not be aware of the data and in order to avoid state space explosion, those control entities are defined only as ids. Depending on the requests, the control component either creates control entities or breaks the request into several actions that might be executed, for example triggering an execution of an algorithm or triggering another request to an external service.

When following the DCA architectural pattern, ASD tool is suited for modeling the Control aspect. This is done by designing the control behavior of the system in a set of ASD interface and design models. Those models are later on checked by the verification tool in ASD:Suite. Finally, the code in C++ is generated out of the verified models.

8.3.1. Domain model

“We distinguish between entities whose creation is directly triggered by control components, and those entities whose creation is triggered solely by algorithmic components. Only in case creation of an entity is triggered by a control components, the control component will be aware of the entity, control components should not be aware of entities that are created by algorithmic components. The algorithmic entities add details, that need not be known to the controllers. This also implies that the entities ids that are passed from control components to algorithmic components are always control entities. Using queries, the algorithms will collect the relevant algorithmic entities.” [16]

Within this project in the domain model the necessary control entities were created, note though that the domain model is not part of the public report.

8.4 Failure handling

In ASD tool, all exceptions that occur in a foreign component are considered failures. These exceptions are either caught by the thread’s or by the client’s exception handler. In the second case, the ASD component is left in an inconsistent state. In the both case however, a recovery is not possible. Because of this reason, it is not advised to use an ASD component to propagate the exceptions from the used components to the higher levels.

The solution to this problem is to use hand-written code to propagate the exceptions to the levels above and not using the component. Nevertheless, if there is a need to communicate the error conditions from the used service to the upper layer than that service must be wrapped and the exception must be converted to either a synchronous reply event or to a notification.

Here we distinguish two types of errors:

- **Functional failure** – known failures that are described in the functional document (EDS)
- **Software failures** – any other failures that can occur in algorithms or other services
ASD control logic should not be aware of the software failures since there is no implication in the behavior, caused by these failures. Note that these failures are not part of this scope.

The first type of failures occur in the used services of the ASD component. The functional failures carry an error message that is important for the client and the client is the one that decides what happens afterwards. When these failures occur, they have an effect on the control logic. It needs to delegate the error message to the client and it should either abort or continue its own action. With this been said, we have a situation where the error condition needs to be propagated to the client but the exception cannot be propagated via the ASD component.

There are multiple ways to approach the solution to this problem. The following approaches were investigated in this project:

- **Approach 1**
  One way to approach the exception handling is to intercept the failures just below and above generated ASD code by creating manually service that would do the interception. This way the exception would be routed around ASD and the ASD models would not be aware of the failure. The control logic would run until competition and finally when the control flow returns to the caller of ASD, the failure will be reintroduced into the control path.

- **Approach 2**
  As stated before, the functional failures are a functional concept and it should be clear which functional failure can be raised by which action. Therefore, each functional failure should be modeled as a notification explicitly in the leaves of ASD. In the rest of the ASD components, functional failures would be propagated by generic “Failure” notifications. If an ASD model that gets a generic notification needs to reason about different types of functional failure modes, it can do so through predicate interfaces.

- **Approach 3**
  Similarly as the second approach the functional failures are failures raised by a particular action/algorithm. With this approach the failure handling mechanism should be developed in the ASD used services containing the algorithms that can raise an exception. This exception is caught by the mechanism and stored, and a “Failure” replay is returned to the ASD model. The model stops its execution and returns to its initial state.

Considering the above mentioned approaches and the time allocated for this feature, the handling of the functional failures in this project is managed in the ASD leaves. If a functional failure is thrown by an algorithm, it is converted in a synchronous replay. This replay needs to be propagate to the ASD client and because of that it needs to be propagated to the upper layers. This is done by triggering a failure notification in the upmost ASD model. The control behavior than stops the current execution flow and returns to the initial state, waiting for the new event.

One drawback of this solution is the explosion of notifications that can happen in higher ASD models. Additionally, a mechanism on top of the generated code needs to be developed in order to translate this failure in ASML error type which at the end should be the one sent to the client. Finally, it is highly recommended that a mechanism for logging these failures is developed. Due to the time constraints this was not developed.
9. Implementation

The last two chapters defined the architectural reasoning and the system design decisions which were the reference approach towards the solution of the problem. This chapter continues with describing how these choices were put together and realized.

9.1 Code Generation

An advantage of the model-driven design is the increase of the level of abstraction in the way of working by realizing the design models into code via code generation. As mentioned, the ASD:Suite was used from modeling the control aspect from the DCA. After the successful verification of the ASD models, the code in C++ was generated using the tool. Having the generated code out of the verified models assures that the code is also verified with the same checks, for example the deadlock check, livelock check, and non-determinism check.

An ASD component that is conflict free and successfully verified, it is ready for code generation. Before generating the code, the settings for creating all ASD components were adjusted.

After adjusting the settings, the code was generated from the ASD models. There are two types of files that were generated for C++, the header and source files of the model. The header files are generated as follows: <interfaceModelName>Interface.h and <designModelName>Component.h. The first header file has to contain at least one application interface and zero or more notification interfaces. It is generated out of the Interface ASD Model. The second, defines an ASD component factory interface with GetInstance(), ReleaseInstance(), and GetAPI() for each application interface. If notifications are present, a RegisterCallback() for each notification interface is also included. This header file is a result of a Design ASD Model. Another result of the design model is the source file that follows the following naming <modelName>Component.cpp. This implementation follows the definition in the header files, based on the state pattern.

9.2 Glue code

Another part of the implementation involves the implementation of the glue code between the control-algorithms and control-data. First the details of how the control is related with the algorithms is described followed by the description of the integration between control and data aspect.

From each leaf interface model a skeleton code is generated, which serves as a starting point for implementing the handwritten code for a Foreign component. The skeleton stub code is compile-able such that an executable can be created. The main purpose of the generated stub code is to give a head start in developing the handwritten code. The generated skeleton stub has in place the infrastructural code and clearly pointed out places for the custom code. ASD:Suite provides two types of stubs, client and used component stub. Figure 13 depicts the generate stub dialog. In this project only used component stubs were generated.
Having two different technologies used for the development of the Control and Data aspects (ASD and ASOME respectively), required some additional manual work in order to connect these two pieces together. Services named Garages were created for this reason and their responsibility was to retrieve and persist data. They were used to provide the data input for the algorithms and store their output.

There were usually two types of application interfaces defined in the leaf interface models. One was the algorithm application interface and the other was the data application interfaces. Each of these interfaces defined all necessary action that need to be executed by the control flow. ■
10. Verification

In order to assure that the system is correct with respect to the defined specification, a formal process of verification was defined. This chapter explains how this process was organized and ran.

10.1 Introduction

Software verification is defined as a discipline of software engineering whose goal is to assure that software fully satisfies all the expected requirements [20]. To prove that the software satisfies the expected requirements is not easy. Because of that there are several methods of verifying a software system and one of these methods is the process of formal verification. Formal verification of a software involves proving that the system satisfies a formal specification of its behavior [19]. In other words, the formal verification has to confirm whether the software is correct with respect to its specification [7]. One approach to prove this is model checking which consists of a systematically exhaustive exploration of the mathematical model (this is possible for finite models) [19]. Since this aspect of the software development is very important, it is used to verify each control component before generating any code from the models.

For this purpose the ASD tool is used, which provides verification of both the interface and design models. What is ensured with the ASD model verification is described in Table 5.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface and design models are complete and well formed.</td>
<td>This ensures that every possible scenario is considered and covered in the models and that all models are free from errors such as range errors, livelocks and deadlocks, that design models are deterministic and that data variables are used properly in the design model.</td>
</tr>
<tr>
<td>The design complies with the service specifications of components it uses.</td>
<td>This means that the design never sends any trigger to a used component unless that used component is in such a state that the trigger is legal and that the design is willing to accept every notification and reply event received from the used component whenever, according to its service specification, the used component is allowed to send it.</td>
</tr>
<tr>
<td>The design together with all of its used components complies with the implemented service specification.</td>
<td>This means that all behavior required by the service specification is implemented by the design and all behavior implemented by the design is allowed by the service specification.</td>
</tr>
</tbody>
</table>

Table 5. ASD verification features [6]

First the interface models need to be verified, since verifying the design models might be meaningless. Model verification of an interface component includes [6][7]:

- Modeling errors
  - Invariant checks
  - Guard completeness
  - State variable ranges
- Livelock checks
• Deadlock checks

Once the interface models are verified, the design models can be verified as well. The verification of the design models checks the following [6][7]:

- Deterministic design
- Modeling errors
  - Invariants
  - Guard completeness
  - State variable ranges
  - Illegal behavior
  - Potential queue problems
- Livelock
- Deadlock
- Interface compliance

10.2 Verification of IVSA control components

In the process of verifying the ASD components first the interface models were verified followed by the design models verification. All of the provided checks from the tool were used in the process, because all of the aspects are important for the process to assure that the components are correct and complete. The deadlock checks helps avoid the situation when the system does not react on any stimulus. The livelock checks on the other hand assures that the component is not busy constantly with its own behavior. This check is important if we want to make sure that the component will not cease to serve its clients.

Before the start of the verification, all the checks were enabled. The most common problems that were arising while developing the design of the IVSA control were the deadlock, livelock and determinism issues. The deterministic checks are extremely important for control logic, since it is very important to have defined deterministic choice in each step where a decision needs to be taken. Note, however that interface models are not required to be deterministic. “This is because an interface model is an abstraction of all possible compliant designs that could implement it.” [6] An example of an output of the verification process is given in Figure 14. In this case the verification result is output of AlarmSystem example from the ASD:Suite.

![Figure 14. Verification results [6]](image-url)
What was different in the verification is that the verification of the interface models was only including the verification for that interface, as where for the design models the verification is running checks on the design model and on the referenced interfaces as well. The assumption here is that the referenced interfaces are implemented by their services correctly and the services behave as specified. In case there was an error while verifying the components, the results were presented using a sequence diagram. An example of such diagram is given in Figure 15 using the AlarmSystem ASD models.

Figure 15. Sequence diagram for failed checks [6]

The sequence diagram shows the model verification errors in terms of the trigger events and the actions executed in particular state. Once the problem was detected, the SBS was updated with the new solution. Sometimes, the new solution was triggering modification on other models and for that reason those models had to be modified and verified again. This process was repeated several times during the design process until the correct and complete design was positively verified. ■
11. Validation

The previous chapter described the process that ensures the system was built correctly. This chapter describes the process of assuring the system that is created is the right system.

11.1 Introduction

After having the basic verification checks, a process is still needed to prove that we built the right system. In order to satisfy this multiple methods for validating a software were used in this project. Validation is the process of evaluating software product, during or at the end of the development process, to determine whether it satisfies the specified requirements. [21] The aim of this development process was to make sure that we built the correct system which complies with the requirements and meets its intent.

Verification helps to determine whether the software is of high quality, but it will not ensure that the system is useful, for that purpose the validation testing methods are needed. Opposite of the verification process, the validation process involves execution of the code. There are multiple methods that can be used to validate a software executing its code, such as unit testing, integration testing, system testing, acceptance testing and regression testing. The following chapters describe which and how some of these methods were used in this project.

11.2 Unit tests

Unit testing is software testing method where individual units of a software are tested. The purpose is to isolate each unit of the software and validate that it performs as designed. Employing this method helps finding the problems early in the development cycle. Another advantage of this method is that it helps reducing the time for integration testing. Unit testing frameworks, drivers, stubs, and mocks are used to assist in unit testing. [22][23]

In order to test ASD components, it is needed to trigger some event on the top the stack of components and inject behavior on the bottom of the stack (ASD leaf nodes). But the main issue is that the ASD generated code is not easy to test, dependency injection is difficult and the code is most of the times build in a static tree of components. For instance, if we want to test the complete ASD stack of components an action in the top most component needs to be triggered and all the leaf components needs to be mocked. Since ASD doesn’t allow dependency injection, the singleton pattern is used in every leaf that behavior needs to be injected. In this way all the behavior variations can be tested using a mocking framework.

In this project Google’s C++ test framework, Google Test (GTest) [24], is used. The unit tests developed for the control aspect make use of the GTest and GMock tools. “Unit tests, test the behavior of a unit such as a class. Modules are tested in isolation using Mock objects (to test the correct sequence of actions) or Fake objects (providing fake data to test on a specific outcome)” [25].
11.3  **Behave tests**

Behavior testing is a testing of the external behavior of a software and it represent in a way a black box testing technique. The purpose of this testing is to test the requirements on a component level. In other words, they are used to test whether the component functionally does what it is supposed to do. [25] Because the behave tests are described on a high level they are easy to understand not just by the software engineers but by all the stakeholders involved in the project.

The behave test are written in Gherkin language. “Gherkin is a business readable, domain specific language created especially for behavior descriptions. It gives you the ability to remove logic details from behavior tests.” [26] In order to be able to test the ASD components with behave tests, the ASD test generation tool is used similarly as for the unit tests. The tool generates the Boost Python wrappers for the test clients and servers.

11.4  **Code review**

In the future, the implementation of the use case is expected to be included in the final production code. Therefore, it was very important that the quality of the code is satisfied. For this reason, a code review was conducted on every implemented feature. This code review was done by the direct supervisors of this project and Project 2, as well as from the trainee working on the Project 2. Only after each of these members approved the implementation, it was merged with the “master” branch. This branch contained the integrated (with Project 2), approved and tested version of the latest version of the component.
12. Deployment

This chapter addresses the deployment of the developed and integrated solution.

12.1 Deployment of the KVIVSA component

The development of this component was done in so called Virtual Desktop Infrastructure (VDI) environment. [27] This environment is already pre-configured and loaded with the necessary tools for software development.

Each IVSA feature that was created was developed on a separate development branch. The versioning control was managed with Git. [28] Using Git, each step of the development was maintained under separate branch. This included separate branches for algorithms (Project 2), and separate branches for the control aspect. After the development of each features was done, the code was reviewed by the directly involved stakeholders of the project. When the review of the code was approved, the feature branch was merged in to the “master” branch which contained the full, reviewed and approved version.

The deliverable was a combination of the work done in this and in Project 2. In order to avoid integration problems in the code from these two project, continuous integration was used. The tool used for this purpose was Jenkins. [29]

12.2 ASD models test deployment strategy

After the code implementation was done, the code was tested on a unit level. For the ASD models the deployment strategy used is that every design model and leaf component is deployed as a library. Libraries are linked together during execution time. In order to test, the libraries that we would like to mock are replaced by a different library for testing. The generated test binary will link the real production libraries against the mocked leaf libraries. This allowed us to test production libraries and inject behavior on the mocked leaf libraries.
13. Project Management

This chapter describes how this project was realized. It addresses the project planning and management process.

13.1 Realization dimension

The realization dimension defines how the system is developed and maintained as well as how the development process is organized. Along this dimension, it is defined clearly who does what and when in order to build the elements from the design and satisfy the defined system functionality. What the realization dimension represents is given on Figure 16. [4]

One of the first steps done in this project was to define the way of working and plan the execution of the project. In this development phase, together with the stakeholders, the work packages were defined and the responsibilities of everybody involved in the project.

Since this project was executed in parallel with Project 2, it was of the highest importance to clearly separate the responsibilities of each of the projects. This decision was agreed in the first two weeks of the project. The scope of the project was defined clearly in terms of the DCA pattern, describing exactly which aspects is this project responsible for.

After distinctly defining the responsibilities between the two projects, the way of working was established. This project management follows the scrum and agile methodologies.[30][31] Following this methodologies the project was executed in iterations,
called sprints. Each sprint planning was done in advance, meaning that each sprint planning was done at the end of the previous sprint. After finishing the sprint, a combined demo presentation with the Project 2 trainee was carried out. The goal of this demo presentation was to update the main stakeholders on the project progress, discuss design opportunities and align on the design decisions.

Additionally to the agile and scrum methodologies methods there was a Project Steering Group (PSG) that was monitoring the execution and the development of the project. This group consisted of the trainee, the TU/e supervisor (Anton Wijs) and the company supervisor (Santiago Cifuentes Costa). PSG meetings were organized throughout this project. The goal of these meetings was to track the progress as well as discuss some issues or challenges during the execution of the project. Additionally, on some of the PSG meetings the trainee’s progress was discussed and a feedback from the supervisors was obtained. These meetings were useful to track the progress, assess the quality of the work and define the course of the next period.

Like mentioned, an extensive planning was done in the beginning of the project and updated accordingly during the nine months period. The following sections elaborate on the project planning and scheduling.

### 13.2 Project Planning and Scheduling

#### 13.2.1. Backlog planning

After coming to an agreement between all interested parties in this project about the way of working, the backlog planning took place. This planning was closely supervised by the direct supervisors from the company. First the epics and the version deliverables were defined. Since in the beginning of the project there were many uncertainties which included lack of domain and software tools knowledge, only the first versions deliverables were defined. After grasping well the domain and tools knowledge, the rest of the version deliverables were defined. After having the backlog stories, each sprint was populated by choosing the estimated stories (with highest priority, and without dependencies) from the backlog.

As mentioned previously the backlog contained the main stories that needed to be executed in this project and the stories for the Project 2. Because these two projects were closely related, the version deliverables defined were combination of the stories done in this and in Project 2. After having the backlog stories, each sprint was populated by choosing the estimated stories (with highest priority, and without dependencies) from the backlog.

The management of the progress and the planning of this project was administered by using the JIRA tool. Using this tool we were able to track the issues that were ongoing, done and the issues that needed to be finished in a defined period of time. Aside of the trainees that were most of the time updating the state of the backlog, the progress was most frequently supervised by the company supervisor. During the implementation and design phases we used to have daily updates (stand-ups). These meetings were useful to keep track on the progress and reveal the challenges that were faced and needed attention.

#### 13.2.2. Gantt chart

Apart from the backlog management an effort was made to maintain a Gantt chart of the project planning. The initial planning for the project schedule was done in the beginning phase of the project. Because at that point not everything was clear and the iterative project management methodology was used, the actual execution differed a lot from the planned from the beginning. Some of the tasks were completely dropped and some were prioritized less by the end of the project; hence there were tasks that were chosen to be developed instead of others in interest of the stakeholders needs. Somewhere in the middle of the project, the Gantt chart was not used anymore. It was used again by the end of the project, when the focus was on writing the documentation.
13.3 **Realization strategy**

A realization strategy was established that was followed in every iteration of the project, depicted on *Figure 17.*

Starting point of each iteration was the planning phase. This phase included choosing an estimated task/story from the backlog and adding it to the upcoming sprint. Usually each sprint took two weeks, but sometimes this time frame was extended. The following sprint was based on the information gathered from the previous sprint such as how many story points were burned in the previous sprint.

After having the schedule and planning for the next (short) period, the design of the chosen stories was constructed. Since using the ASD it was possible to verify the design before implementing any code, the next phase was the formal verification. If the verification was successful, the code was generated from the verified models. In this, implementation phase the handwritten code was added in order to integrate with the Data and Algorithms.

Next phase was the validation phase. In this phase each task was validated using different methods. One method was the unit and behavior testing and the other validation method included code review. At this point the developed feature was available for the supervisors to review and give feedback. In case of a feedback, the implementation of the tasks was updated accordingly.

Finally, after finishing all of these phases the next cycle of the same phases was begun.
14. Project Retrospective

This final chapter revisits the design opportunities observed in the section 2.4. Finally, based on the author’s viewpoint a reflection on how the project was carried out is presented.

14.1 Reflection

The execution of this ten months project was a great experience both personally and professionally. I had the opportunity to learn from a number of great experts in the domain. With this project I experienced a lot of new technologies and methodologies, as well as number of challenges.

First challenge that was faced in this project was the domain and the context of the problem that this project was required to solve. This was the first time to face the ASML’s domain and the discipline that they were working with. Because the domain knowledge regarding the TWINSCAN machines is quite broad, it was really hard to limit the domain of the problem to an acceptable and feasible extent for this ten months period. But with the help of the supervisors and the experts in the domain, this was done early in the beginning of the project.

Another challenge was posed by the new technologies that were used in the execution of the project. I already had knowledge of model-driven development and some experience with the programming language. Even though, this was not enough for the development. I had to dedicate a certain amount of time to explore the new technologies (ASOME and ASD). However, I have to stress that without the help of my supervisor I would not have been able to grasp the needed knowledge so fast.

Designing the solution also had a huge impact on developing my technical designer skills. By using trial and error, I learned that trying to design the first model from the first try should not be the goal. Instead, by dividing the problem in smaller steps, it is easier to design and build the solution. One gain is that you learn from the mistakes and another is that you can be more effective in the next iteration.

The frequent reviews of my work gave me meaningful feedback which contributed with my professional growth. Same goes for the frequent demonstration meetings which provided experience in this area as well as in managing people’s expectations. I was able to gain a considerable experience in organizing work tasks and managing the project. Another aspect which I got to practice was the team/stakeholder communication, which was not on a high level especially in the beginning.

To summarize, this project was a valuable experience in terms of developing my professional skills as well as achieving to satisfy stakeholders needs.

14.2 Design opportunities revisited

As mentioned in section 2.4, there were three design opportunities that were recognized in this project, methodical approach, functionality and complexity. Here we revisit those design opportunities.

- **Methodical approach:** In this project different methodologies were used while designing the deliverables. Model-driven tools and technologies were
used for both the data and control aspects. Additionally, an architectural pattern was used to separate different system aspects (DCA). Further, formal techniques for checking the requirements that needed to be fulfilled, were defined and followed.

- **Functionality:** By establishing well defined verification and validation processes in order to check whether the requirements were met, this design opportunity was accessed successfully at the end. Another factor that proved the functionality of the artifact is the satisfied stakeholders concerns (checked by the frequent stakeholders reviews). Note, that we are referring here only to the supervisors responsible for maintaining the deliverables after the end of this project.

- **Complexity:** Since the problem required knowledge of methods and techniques from a various disciplines, a “functional” design was prepared by physics experts. This design was a base ground for the software artefact. Without this input, this project would not have been possible.

### 14.3 Process

One drawback on the management process was that some of the tasks were executed longer than planned, which was somewhat result of not so good team communication. This especially holds because I was not able to concisely formulate my statements, which hindered the effectiveness of the discussion and the overall performance. Another point that was made by my supervisors was that I did not reach for assistance on time. This resulted in working longer than anticipated on the tasks, trying to find the solution by myself instead of asking for help from the experts. Since this was noticed in the beginning of the project, I was working on the issue throughout the execution of the project.

Overall, when doing the retrospective on the process of managing this project, it was concluded that the managing, organizing and the task handling were carried out on a high level. The continuous retrospective and feedback sessions with the supervisors helped steering this project towards successful execution. Additionally, the methods used in section 13, proved to be useful and effective in supporting the delivery of the satisfactory outcome.
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chuck</strong></td>
<td>In ASML terminology, a chuck is generally referred to as a physical unit inside a lithography system that is used to 'temporarily' hold materials, like a wafer or a reticle, for a specific purpose.</td>
</tr>
<tr>
<td><strong>Wafer</strong></td>
<td>A wafer is a circular slice of semi-conducting material that serves as a substrate for the manufacturing of chips.</td>
</tr>
<tr>
<td><strong>Reticle</strong></td>
<td>A reticle or photomask is a carrier with the negative or positive version of a (part of a) pattern of a layer of a microlithographic device.</td>
</tr>
<tr>
<td><strong>Exposure</strong></td>
<td>The generic ASML term Exposure is used for referring to the process of controlled exposing of wafers coated with a photo sensitive layer (resist) to actinic light resulting from the source.</td>
</tr>
<tr>
<td><strong>EDS</strong></td>
<td>Element Design Specification is a document used within ASML to describe the Design/Solution of a (sub)system.</td>
</tr>
<tr>
<td><strong>ASD:Suite</strong></td>
<td>ASD stands for Analytical Software Design, which is an approach to create complete and mathematically proven software designs.</td>
</tr>
<tr>
<td><strong>ASOME</strong></td>
<td>Data tool suite</td>
</tr>
<tr>
<td><strong>Measurement results</strong></td>
<td>Measurement results (for Leveling) give the vertical position of the plate surface in PCS at one or more positions on the plate. In practice, they contain X and Y positions and the corresponding Z positions in PCS. In the case of the capture measurement result, also an Ry position in PCS is included to indicate the plate tilt at that location. Measurement results are the result of modeling these scan results.</td>
</tr>
<tr>
<td><strong>Scan results</strong></td>
<td>Scan results are the raw results as obtained from the LS driver; Scan results contain no horizontal information.</td>
</tr>
<tr>
<td><strong>Measure Sequence</strong></td>
<td>Measure Sequence controls scheduling of all measurement actions at the measure side.</td>
</tr>
<tr>
<td><strong>XT</strong></td>
<td>Generation of the ASML TWINSCAN machine. The system is a high-productivity, dual-stage ArF lithography tool designed for volume 300-mm wafer production at 65-nm resolution.</td>
</tr>
<tr>
<td><strong>NXT</strong></td>
<td>Successor of the ASML XT TWINSCAN machine. The system is a high-productivity, dual-stage immersion lithography tool designed for volume production 300-mm wafers at the sub 10-nm node.</td>
</tr>
<tr>
<td><strong>NXE</strong></td>
<td>Successor of the ASML NXT TWINSCAN machine. NXE refers to the first EUV-Platform lithography systems.</td>
</tr>
</tbody>
</table>
References


About the Authors

Anita Asprovska received her Bachelor’s degree at the Faculty of Computer Science and Engineering from the Ss. Cyril and Methodius University, Skopje, Macedonia in May 2014. She conducted her master studies in Software Engineering TEMPUS, at the same university, with focus on analyzing, designing and documenting appropriate solutions in more than one application domains using SE approaches that integrate ethical, social, legal and economic concerns. During her master studies she was working in an international company “Singular”, as Java Enterprise Developer. While working there she participated in developing web applications for online gambling.
Where innovation starts