Wafer flow simulator visualizer

Citation for published version (APA):

Document status and date:
Published: 01/01/2013

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: 15. Jan. 2020
Wafer Flow Simulator Visualizer

Umut Uyumaz
September 2013
Wafer Flow Simulator Visualizer

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

ASML Netherlands B.V.

Steering Group

Umut Uyumaz
Ernest Mithun Xavier Lobo
Istvan Nagy
Huub van de Wetering

Date

September 2013
Abstract
The control software that drives the ASML photolithography systems is tested on different levels ranging from using simulators to using real hardware. The wafer handling (WH) subsystem uses the Wafer Flow Simulator (WFS) for testing. However, the WFS also requires testing and its information polluted, big log files are cumbersome to debug. Moreover, the interior of the photolithography systems cannot be observed during the production. This project proposes a 3D visualizer to make the testing process easier for the WFS and to enable the observation of the WH subsystem. The visualizer puts all the information in the WFS (e.g., position, orientation of the peripherals) in one 3D scene that otherwise the user would have to check multiple (in order of ten to fifteen) different windows to see the same information. The visualization tool can also indicate the non-nominal behaviors that are injected by the user and can read and re-play the system trace files. A prototype of this tool was already used to observe the WH of an existing system. Moreover, the visualization tool can be used to visualize the WH subsystem of the machines that are still being designed and to train new employees that will work on it.

Keywords
3D visualization, information visualization, simulation, photolithography, material flow

Preferred reference
Foreword

Software-in-the-loop simulation plays an important role in the testing and qualification of complex high-tech systems such as ASML’s wafer scanners. ASML uses a domain specific modeling language (DSL) and model-driven engineering (MDE) environment to model and generate software-in-the-loop simulators for the simulation of wafer flow in ASML’s wafer scanners. The current simulation uses textual tracing to show simulation states during execution.

In this thesis, Umut describes a 3D visualization extension to the simulation with minimal impact on the current simulator. In addition to the visualizer, he also describes a timing library that can be used to provide accurate timings to actualizations for more realistic visualizations. The implemented visualizer designed and developed by Umut has already proven to be valuable in the software development of new ASML products where the yet-to-be-manufactured hardware can already be virtually visualized early in the development cycle.

Ernest Mithun Xavier Lobo
Software Design Engineer, ASML
Eindhoven, 16th September 2013
Preface

This thesis represents the results of the graduation project of Software Technology post-master program at Stan Ackermans Institute. The project is held by the supervision of ASML. This thesis focuses on software design aspects. It describes the problems and gives a detailed design about the solution. Requirement engineering related subjects can be seen in Chapter 4 and software design related subjects can be seen in Chapter 5.

September 2013
Acknowledgements

I would like to express my very great appreciation to my company supervisors, Ernest Mithun Xavier Lobo and Istvan Nagy for their continuous support, guidance and feedback throughout the project. Their experience helped me to grasp the general and technical environment at ASML while their enthusiasm kept me highly motivated. I would also thank the Wafer Handler team members, Sander van Woensel, Berrie van den Eijnden, Dennis Verhaegh, and Ge van Lier for their technical inputs and feedbacks. I am grateful to my university supervisor, Dr. Huub van de Wetering for assessing my work, and for being an important part of my project steering group.

I would like to extend my thanks to the program director of PDEng Software Technology, Dr. Ad Aerts, for his support and management of the entire curriculum of the Software Technology program. I would like to thank all the coaches for their instructive lectures which helped me greatly improve myself during last two years of studies. Special thanks to the management assistant, Maggy de Wert for taking great care of all the trainees with so much love and enthusiasm.

I would like to thank all my colleagues for their feedback and support throughout the program. The pleasure working with them in the OOTI office kept me pushing and improving myself towards to be a better designer.

Finally, I would like to offer my special thanks to my family, especially to Begüm Erten Uyumaz, my parents, my brother, and my friends for their love, support, and encouragement.

September 2013
Executive Summary

ASML provides semiconductor manufacturers with advanced lithography systems. The complex control software that drives these ASML lithography systems is tested on different levels ranging from using simulators to using real hardware. The wafer handling (WH) subsystem uses the Wafer Flow Simulator (WFS) for its testing among others as well. The WFS was introduced to do as much testing as early as possible even when HW is not available yet.

The WH subsystem is shielded and the peripherals in it cannot be observed during the production or during the machine recovery. In this project, we aim to improve the awareness on the peripherals and the wafers located in the wafer handler subsystem and make system trace file analysis quicker and more intuitive by using a 3D visualization.

With the 3D visualizer tool, the following items are delivered:

Visualization Software: With the visualization software (on Windows), the user is able to capture the internal states of the peripherals, the wafers, and the sensors in the WFS. Having all the information in a 3D scene is valuable in terms of understanding how the wafer handler works. The visual information provided by the WFSVis can be also used to debug the WFS functionality. Moreover, the WFSVis can also generate 3D scenes for system configuration that are still under design, by using their model specifications and this 3D scene gives information about the system to the developers who haven't seen its layout.

Visualization Code Generator: The code generator is extended to produce visualization functionality. With this feature, a visualization tool can be generated for any of the system configurations. This is important for ASML since there are different projects with different configurations running in parallel.

Log File Parser: The parser tool uses the log files are produced by the TWINSCAN software. The offline visualization of the log files makes debugging easier since the user can follow the actuations in a 3D scene instead of going through thousands lines of text.

The visualization tool is already used in developing and executing new test cases for the WFS. It also received good feedback from the developers of the QXE system which was still under design at the time this project was developed. The QXE system uses a wafer handler subsystem with different layout and the visualization tool was able to show it in a 3D scene using its model file.
Table of Contents

Foreword........................................................................................................... v
Preface........................................................................................................... vii
Acknowledgements ....................................................................................... ix
Executive Summary ..................................................................................... xi
List of Figures............................................................................................... xv
List of Tables................................................................................................. xvii

1. Introduction ............................................................................................. 1
   1.1 Photolithography.................................................................................. 1
   1.2 ASML’s Photolithography System..................................................... 3
       1.2.1 Wafer Handler and Stage............................................................. 3
       1.2.2 Reticle Handler and Stage......................................................... 4
       1.2.3 Illumination and Projection....................................................... 4
   1.3 Software in a Loop Simulator.............................................................. 4
   1.4 Wafer Handler Limitations.............................................................. 5
   1.5 Outline............................................................................................... 5

2. Problem Analysis...................................................................................... 7
   2.1 Problem Description........................................................................... 7
   2.2 Envisioned Solution........................................................................... 7
   2.3 Project Scope...................................................................................... 8
   2.4 Stakeholder Analysis.......................................................................... 8

3. Domain Analysis....................................................................................... 9
   3.1 The Wafer Handler Subsystem......................................................... 9
   3.2 The Wafer Flow Simulator............................................................... 9
   3.3 Visualization....................................................................................... 11

4. System Requirements............................................................................... 13
   4.1 High Level Requirements............................................................... 13
   4.2 Functional Requirements................................................................ 13
       4.2.1 Visualize Wafers......................................................................... 13
       4.2.2 Visualize Pedestals................................................................... 13
       4.2.3 Visualize Actuators................................................................... 14
       4.2.4 Visualize Light Beam Sensors.................................................... 14
       4.2.5 Visualize Wafer Transfers.......................................................... 14
       4.2.6 Visualize Transfer Areas............................................................ 14
       4.2.7 Visualize Error Injections.......................................................... 15
   4.3 Non-Functional Requirements......................................................... 15
List of Figures

Figure 1-1 Stone lithography pattern of an old map of Munich and its imprinted paper ............................................................................................................ 1
Figure 1-2 Photolithography imprint ........................................................................................................ 2
Figure 1-3 Photolithography workflow .................................................................................................... 3
Figure 1-4 Subsystems of the ASML’s TWINSCAN machine ............................................................... 3
Figure 1-5 Overview of the WFS component and its interaction with the control software................................. 5
Figure 3-1 Overview of the wafer handler subsystem ....................................................................... 9
Figure 3-2 Wafer transfer between two transfer areas .................................................................. 10
Figure 3-3 Wafer light beam sensor interaction (Ernest’s XPO 2011 Presentation) ......................... 10
Figure 3-4 Wafer displacement injection .................................................................................. 11
Figure 5-1 4+1 Architectural View .................................................................................................. 18
Figure 5-2 WFSVis Context Diagram ............................................................................................. 18
Figure 5-3 Visualize Wafer Handler use case .................................................................................. 19
Figure 5-4 Error injection use case .................................................................................................. 20
Figure 5-5 Visualize trace file use case ........................................................................................... 20
Figure 5-6 WFS Structure ................................................................................................................. 21
Figure 5-7 Visualizer Structure ......................................................................................................... 22
Figure 5-8 Visualizer Node Structure ............................................................................................... 22
Figure 5-9 Visualizer Command Structure .......................................................................................... 23
Figure 5-10 Command Implementation Structure ............................................................................... 24
Figure 5-11 The Bridge Pattern ......................................................................................................... 25
Figure 6-1 Milestone deadlines ............................................................................................................ 27
Figure 6-2 Work Breakdown Structure of the WFSVis .................................................................. 28
Figure 6-3 Risk impact vs. risk probability chart ............................................................................... 31
List of Tables

[All tables in the report should be numbered and should have an appropriate title.]

Table 1 Organizational Stakeholders ................................................................. 8
Table 2 Technical Stakeholders ........................................................................ 8
Table 3 Visualize Wafer Requirement ............................................................... 13
Table 4 Visualize Pedestal Requirement ............................................................ 14
Table 5 Visualize Actuators Requirement .......................................................... 14
Table 6 Visualize Light Beam Sensors Requirement .......................................... 14
Table 7 Visualize Wafer Transfers Requirement ............................................... 14
Table 8 Visualize Wafer Transfers Requirement ............................................... 15
Table 9 Visualize Wafer Transfers Requirement ............................................... 15
Table 10 Non-functional requirements of WFSVis ........................................... 15
Table 11 Milestones and Task Distribution ....................................................... 29
Table 12 The risk items involved with the WFSVIS project ............................. 30
1. Introduction

ASML designs, develops, integrates and services advanced photolithography\(^1\) systems to produce semiconductors. ASML makes the machines that print the microchips that are used in almost every technological product such as computers, cell phones and washing machines \([1]\). It is a global company founded in 1984 and headquartered in Veldhoven, the Netherlands. The main drive of the company is to increase the throughput of its clients by decreasing the size of microchips and increasing the production speed. ASML is one of the key actors that have enabled the industry to keep up to speed with Moore's Law\(^2\) in last two decades.

The next section gives a brief description about the photolithography process. In Section 1.2 the photolithography system of ASML is explained briefly. Section 1.3 discusses the WFS. Next, in Section 1.4, current limitation is given. Finally, Section 1.5 gives an overview of document structure.

1.1 Photolithography

The term photolithography comes from its Greek origin “litho -graphy” which means stone-writing. In lithography a stone pattern is used to imprint the pattern on a piece of paper. In Figure 1-1, an old regional map of Munich is imprinted on a paper using lithography technique. This technique was used in the nineteenth century as a cheap method of publishing theatrical works \([2]\).

![Figure 1-1 Stone lithography pattern of an old map of Munich and its imprinted paper.](image)

Photolithography has similar principles to lithography. In photolithography there is a pattern to be imprinted as well but the difference arises in how this pattern is imprinted and on what kind of substrate. As the prefix photo suggests, it uses a light source to imprint a pattern, similar to how photo cameras work. Since it uses light, its substrate needs to be a material that is sensitive to its light source (photoresist). In Figure 1-2 there is an illustration of photolithography imprinting. A light is illuminated from the light source and goes through a lens to have a focused image on the substrate. The substrate is placed above the bottom. In between the substrate and the lens, the pattern to be imprinted is placed. Light goes through holes in this pattern and reaches

---

\(^1\) Photolithography is a process used in microfabrication to pattern parts of a thin film or the bulk of a substrate.

\(^2\) Moore's Law is a computing term which originated around 1970; the simplified version of this law states that processor speeds, or overall processing power for computers will double every two years.
the substrate. Since we have a photoresist substrate, the pattern is imprinted after a chemical reaction in the photoresist substrate.

Figure 1-2 Photolithography imprint

ASML produces complex photolithography systems that are capable of imprinting nanoscale\(^3\) (down to 22nm) patterns for microchip manufacturers. In chip manufacturing there are many more steps than photolithographic imprinting. The chip manufacturing process can be seen in Figure 1-3. Imprinting happens in the exposure part (red arrow). In the preparation part, first a silicon cylinder is sliced into thin disks that are called wafers. Then these wafers are polished and a conductive material is put on them. Then they are covered with a photoresist material to get ready for photolithographic imprinting. The imprinting is done at this step. After the imprinting, there are several chemical steps to make the printed pattern permanent on the wafer. The cycle in photolithography workflow can repeat itself several times to build multi-layered integrated circuits on a microchip.

\(^3\) The human hair growth rate can be taken as an example to illustrate how small this scale is. A human hair grows approximately 5 nm each second [3].
1.2 ASML’s Photolithography System

As previously mentioned, ASML’s lithography machines perform the exposure (imprinting) stage of the photolithography process. The throughput of these machines is important to ASML’s customers and the required productivity (number of wafers processed correctly in an hour) heavily depends on imaging quality (resolution) and overlay accuracy (accuracy of overlapping layers) of the system. Such a complex lithography machine is divided into several subsystems to increase the accuracy and the productivity of the main system. In Figure 1-4, these subsystems for the TWINSCAN machine are shown. A description for each subsystem is given in the following subsections.

1.2.1 Wafer Handler and Stage

The wafer handler is the module where the wafer enters and leaves the lithography system. Its main responsibility is to pre-align the wafer before the wafer can be delivered to the wafer stage.

In ASML terminology, a stage is part of a lithography system. It receives and returns materials from and to a handler and accurately moves and positions these materials in relation to the lens for the exposure process. The Wafer Stage does so with wafers, the Reticle Stage with reticles. In ASML’s lithography machines, the wafer stage subsystem is the machine module responsible for carrying and positioning the wafer.
during exposure and measurement (for alignment). In a TWINSCAN machine, the wafer stage can handle two wafers in parallel. The name TWINSCAN comes from this feature.

### 1.2.2. Reticle Handler and Stage

In ASML's lithography systems the pattern to be imprinted is called a reticle (or mask). A reticle handler subsystem takes a reticle from a reticle-pod where reticles are being held and places it accurately on the reticle stage.

The reticle stage subsystem is responsible for moving the reticle accurately during exposure in coordination with the wafer stage subsystem.

### 1.2.3. Illumination and Projection

The illumination subsystem provides a light with the right properties to illuminate the reticle.

The projection subsystem is responsible for aligning the lens to project the reticle onto (part of) the wafer. The wafer stage, the reticle stage and the projection subsystem all work in coordination to have the best accuracy on an imprinted wafer.

### 1.3 Software in a Loop Simulator

The more complex the system is, the harder it becomes to control it. Beyond some complexity, a system becomes too hard to control. Computers and software play a crucial role to drive such complex systems. ASML's lithography machine is one of these complex systems that are controlled by software components. The complex hardware parts are driven by complex controller software. The system is so complex that the software has more than 35 million lines of code\(^4\). ASML's controller software is as complex as a computer operating system.\(^4\) Since each machine is extremely expensive and complex to construct, the reliability of both software and hardware is an important issue. In order to minimize the risk of damage to the hardware as well as to the system operators, safety checks play an important role in the software development. Moreover, the software and the hardware must work together such that the performance requirements of the machine are realized.

Effective and efficient testing of the software is an absolute necessity to facilitate this reliability and throughput. This is an important challenge in ASML's software development framework. Since testing on the real machine is time consuming and expensive in terms of availability, cost, and risk, there are only a few machines in ASML for development. Many developers need to test their code on these machines. Therefore availability is a big problem for testing. The machines are expensive (to the order of millions of dollars each) and testing software brings risks to damage them.

One can use software-in-the-loop (SIL) simulations to eliminate the need of hardware during testing. SIL simulations behave as the real hardware. SIL simulation gets the function calls from the control software and replies back to it with the hardware sensor values and positions, based on consistent simulated system state (see Figure 1-5). As for the control software perspective, it is no different than communication to real hardware.

To increase availability and decrease the cost and risk ASML developed a Wafer Flow Simulator (WFS) as a SIL simulator. This simulator covers the wafer handling subsystem and simulates both nominal and non-nominal behavior of the real machine. Non-nominal simulation is achieved by having an error/fault injection mechanism in the simulator\(^5\). This enables one to test the software on the simulator as if it were running on the real machine and to use time efficiently.

---

\(^4\) Windows XP is compiled from 45 million lines of code\(^4\)
1.4 Wafer Handler Limitations

There is a limited observation on the TWINSCAN machine and on the wafer handler subsystem in general during production. The exterior panels of the TWINSCAN machine cannot be opened while the machine is in production. Therefore, the log files that are produced by the TWINSCAN machine are used in the wafer handler subsystem analysis. However, these log files are quite long files and analyzing them is cumbersome.

1.5 Outline

This report describes the development of the WFS Visualizer (WFSVis). In Chapter 2, an analysis of the problem domain is made and problem statements and stakeholder analysis are given. In Chapter 3 analysis of the wafer handler subsystem, the WFS are given. Next, the requirements of the WFSVis are listed. In Chapter 5, the design is explained in detail. Chapter 6 gives an overview on the project management. In Chapter 7, the conclusion and future directions are given. Lastly, in Chapter 8, a reflection on the project is given.
2. Problem Analysis

In this chapter, the limitations listed in Section 1.4 are explained in more detail to give a deeper understanding of the problems and their causes. Firstly, in Section 2.1 the problem statements are listed. Next, the initial envisioned solution, project deliverables and project scope are given. Lastly, the stakeholders are analyzed in Section 2.4.

2.1 Problem Description

As described in Section 1.4 the limitation of the wafer handler subsystem is that it is shielded and cannot be observed from outside during production. To make the wafer handler subsystem more visible to outside world, a logging mechanism is implemented to record each actuation and state of the software into files. These files are analyzed afterwards to understand execution process and the bugs within the software. However, these files are large log files that require deep analysis to understand. This makes analysis cumbersome in four different domains:

1. Wafer Handler Subsystem: The wafer handler subsystem is shielded in the TWINSCAN system and cannot be observer during the production. The log files that are produced are large and cumbersome to analyze manually, therefore this creates an extra downtime for the machines during error/bug analysis which increases the costs for the client.

2. Wafer Handler Recovery Sequences: Recovery scenarios are used to correct the TWINSCAN system from an erroneous state. It is hard to analyze recovery sequences as the difficult-to-read log files with thousands of lines are the only source. The engineers who try to diagnose an erroneous situation and recover the machine need to analyze these large log files. Again, the inefficient analysis creates a longer downtime and thereby becomes more expensive for the client.

3. Wafer Flow Simulator: The SIL simulator for the wafer handler subsystem, WFS, simulates actuations and sensor changes in an instantaneous and blocking manner. During execution, the simulator produces simulator state dumps after each actuation or state change. Similar to the previous problem descriptions, the state dumps consist of thousands of lines and it is difficult to analyze them to debug the simulator.

2.2 Envisioned Solution

The problems described in the previous section are primarily because of a tedious analysis of textual information. In this section, we propose a solution to represent the same information in a better, more visual and intuitive way. The envisioned solution is a 3D visualizer which will perform the following:

- Visualize the current state of peripherals and interactions, the wafer properties and injected faults in the WFS during production. *(improved observation on the wafer handler subsystem)*
- Visualize history of system states by parsing and re-playing log files *(increased observation on error recovery scenarios)*
Project Deliverables
The deliverables for this project are listed below:

- Visualization Engine that observes the WFS
- Plugin for parsing log files and re-playing them with the WFS
- Documents
  - Product Vision
  - Design Document
  - Presentation and demonstrations
  - Final Technical Report

2.3 Project Scope
This section defines the scope of the project. Below there is a list of items related to scope of the project:

- Project duration is nine months
- WFSVis works on the NXE prototype platform
- WFSVis visualizes only the wafer handling subsystem
- WFSVis does not have detailed meshes for the system peripherals
- WFSVis does not detect collisions
- WFSVis does not detect erroneous machine states

2.4 Stakeholder Analysis
This section lists the stakeholders of the WFSVis.

Table 1 Organizational Stakeholders

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ad Aerts</td>
<td>Program Director of PDEng in Software Technology</td>
<td>Ensure that the project meets the quality requirements of the PDEng programme.</td>
</tr>
<tr>
<td>Huub van de Wetering</td>
<td>University supervisor</td>
<td>Ensure that project is in the right direction and its quality is within the expectation of PDEng programme.</td>
</tr>
<tr>
<td>Rogier Wester</td>
<td>Manager of SW AI Architecture department</td>
<td>Ensure that project brings added value to the company.</td>
</tr>
</tbody>
</table>

Table 2 Technical Stakeholders

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ernest Mithun Xavier Lobo</td>
<td>ASML supervisor from the Architecture and Platform (A&amp;P) Department.</td>
<td>Key role in providing the relevant information required for realizing the project.</td>
</tr>
<tr>
<td>Istvan Nagy</td>
<td>ASML supervisor from the Architecture and Platform (A&amp;P) Department.</td>
<td>Key role in providing the relevant information required for realizing the project.</td>
</tr>
<tr>
<td>Umut Uyumaz</td>
<td>PDEng in Software Technology trainee</td>
<td>Coordinate, design, and develop the project. Ensure that the project results meet the company and the university standards.</td>
</tr>
<tr>
<td>Wafer Handler Team</td>
<td>The development team for the wafer handler subsystem</td>
<td>Provide requirements for the visualization tool. Ensure that the results meet the requirements.</td>
</tr>
</tbody>
</table>
3. Domain Analysis

The WFSVis is an extension to the WFS which simulates wafer flow in the Wafer Handler subsystem. This chapter first gives brief information about the wafer handler subsystem and the modules in it. In addition, in Section 3.2, a detailed explanation of the WFS implementation is given.

3.1 The Wafer Handler Subsystem

The wafer handler subsystem is responsible for wafer flow into and out of the machine. It consists of different modules which are composed of actuators and sensors. In Figure 3-1, shows an overview of the wafer handler subsystem. The subsystem consists of an atmospheric side (which is indicated with light blue color) and a vacuum side. An adjacent submodule, called track (TR), is responsible for delivering wafers to the pre-alignment unit (PA) of the wafer handler. The PA centers the eccentric wafers and determines the wafer orientation. Since we have an atmospheric side and vacuum side, there is a need for a mechanism to transfer wafers from one side to the other. Two load-locks (LL-I and LL-O) are designed for this purpose. LL-I allows wafers from atmospheric side to be transferred to the vacuum side by pressuring down its chamber to vacuum and LL-O does the opposite by venting air. Each load lock unit has a gate on the atmospheric side and on the vacuum side. In addition, there are light beam sensors (indicated with blue dots) to check wafer presence and wafer positioning. The mechanical robots (LR, UR, and IVR) are responsible for carrying the wafer between modules in the wafer handler subsystem. The load robot (LR) transports wafers from PA to LL-I and unload robot transports wafers from LL-O to the discharge unit (DU). Then, from the DU, the wafer is picked up by the TR [6].

![Figure 3-1 Overview of the wafer handler subsystem](image)

3.2 The Wafer Flow Simulator

WFS is the component where the hardware submodules described in the previous section are simulated in the software. It has software components for each of the peripherals that model the composition and behavior of the submodules of WH. It uses geometry to check interactions between peripherals (e.g., interaction between a wafer and the load robot). Moreover, it simulates the environment, and also the material (wafer). It can also simulate non-nominal behaviors using error injections such as wafer loss, wafer displacement or incomplete robot move.
WFS is capable of simulating wafer transfers and sensor interactions. A virtual concept called *transfer area* is created to make a mathematical model of wafer transfer. The transfer areas are attached to holding peripherals (robots, pre-aligners, etc.). If two transfer areas intersect with each other it is assumed that the wafer is transferred between them. In Figure 3-2, a wafer transfer between two peripherals is illustrated. When the robot moves down, the transfer areas of robot and pedestal intersect with each other and the wafer is transferred from the robot to the pedestal.

![Figure 3-2 Wafer transfer between two transfer areas](image)

The actuations can also change the state of a sensor. Figure 3-3 shows an example of a state change on a light sensor. When the robot moves closer to the light beam sensor, the wafer comes in between the light source and the light sensor which causes an interruption.

![Figure 3-3 Wafer light beam sensor interaction](image)

WFS can also simulate non-nominal behaviors in the wafer handler subsystem. An example of such a non-nominal behavior is a wafer displacement scenario. When a
wafer transfer occurs, the WFS can inject a displacement error on the wafer. In Figure 3-4, an example of a displacement injection can be seen. At the right side, the wafer is displaced and a position sensor is interrupted.

![Figure 3-4 Wafer displacement injection](image)

### 3.3 Visualization

As described in the previous sections, the WFS can simulate the submodules in the wafer handling subsystem. It is a textual simulator which writes its states in a state dump file. WFS is being used to test the software without the need of the actual hardware. However, it does not provide us a visual feedback about the wafer handler subsystem. In order to understand the wafer or a submodule state, the analysis of large log files is needed which is cumbersome. The visualization of this virtual hardware can help us to improve the efficiency in log file analysis.
4. System Requirements

In this chapter, the system requirements of the WFSVis are analyzed. These fall into two categories: functional and non-functional. Functional requirements describe a system with respect to its behavior. It specifies what input is expected from the system, how the system behaves under this input and how the output is formed based on the behavior of the system. On the other hand, non-functional requirements specify the non-behavioral criterion that shapes the system according to user's needs (e.g., reliability of a system).

The functional requirements are listed in Section 4.1. Next, non-functional requirements are given in Section 4.3. Lastly, the chosen design criteria for this project are explained in Section 4.4.

4.1 High Level Requirements

This section defines and describes the high level features of the WFS Visualizer. These are the high-level capabilities of the system that are necessary to deliver the following described benefits to the users. The high level requirements are:

- **Visualize WH**: The WFSVis must visualize the wafer handler subsystem and its submodules. The wafer flow must be clearly visible.

- **Visualize Trace**: The WFSVis must parse a trace file and re-play it

4.2 Functional Requirements

In this section, the functional requirements of the WFSVis are given.

4.2.1. Visualize Wafers

Wafer is the main transportable material in the TWINSCAN software and in the WFS as well. The other peripherals are designed to move the wafer within the system. Visualization of wafer characteristics is important because they might change during their transportation from the track to the stage and they affect the quality of imprinting. The wafer characteristics are:

- wafer position
- wafer notch orientation
- wafer radius
- wafer presence on a peripheral

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-VIS-01</td>
<td>All the wafers in the system must be visualized. In addition, their position information, notch orientation, radius and presence on a specific peripheral must also be visible</td>
<td>MUST</td>
</tr>
</tbody>
</table>

4.2.2. Visualize Pedestals
Pedestals are the fixed locations that can hold a wafer. They need to be visible because they are involved in the wafer flow. The visualization of the pedestals also helps the user to understand the 3D scene and machine formation.

Table 4 Visualize Pedestal Requirement

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-VIS-02</td>
<td>All the pedestals in the system and their positions must be visualized.</td>
<td>MUST</td>
</tr>
</tbody>
</table>

4.2.3. **Visualize Actuators**

Actuators play an important role in the WFS. They are responsible for carrying and positioning the wafers. Therefore, it is important to see how they manipulate the wafers.

Table 5 Visualize Actuators Requirement

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-VIS-03</td>
<td>All the actuators in the system, their positions and orientations must be visualized.</td>
<td>MUST</td>
</tr>
</tbody>
</table>

4.2.4. **Visualize Light Beam Sensors**

Light beam sensors are used to see whether a wafer is present on a pedestal (e.g., load lock) and whether it is positioned correctly. Visualizing the state of the light beam sensors is important since the wafer positioning can be inferred by their states and wafer positioning is a must have requirement.

Table 6 Visualize Light Beam Sensors Requirement

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-VIS-04</td>
<td>All the light beam sensors in the system and their states must be visualized.</td>
<td>MUST</td>
</tr>
</tbody>
</table>

4.2.5. **Visualize Wafer Transfers**

A wafer transfer event is a simulator concept and it reflects the physical transfers that are happening in the system. A wafer moves between peripherals via this concept. Therefore, it is important to visualize wafer transfers.

Table 7 Visualize Wafer Transfers Requirement

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-VIS-05</td>
<td>Wafer transfers must be visualized.</td>
<td>MUST</td>
</tr>
</tbody>
</table>

4.2.6. **Visualize Transfer Areas**

A transfer area is a virtual concept that the simulator uses to calculate wafer transfers based on the actuation of the robots. They are important to the WFS developer as they can be used effectively in the WFS debugging.
Table 8 Visualize Wafer Transfers Requirement

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-VIS-06</td>
<td>Transfer areas must be visualized.</td>
<td>MUST</td>
</tr>
</tbody>
</table>

4.2.7. Visualize Error Injections

Virtual errors can be injected to the WFS and they can affect the position of wafer and/or robots (e.g. a wafer can be displaced). It is important for the user to know that the displacement made on a wafer is caused by an injected error in order to avoid any confusion.

Table 9 Visualize Wafer Transfers Requirement

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-VIS-07</td>
<td>When there is an error injected in the WFS, it must be clearly visible to the user.</td>
<td>MUST</td>
</tr>
</tbody>
</table>

4.3 Non-Functional Requirements

Non-functional requirements of the WFSVis are listed in Table 10.

Table 10 Non-functional requirements of WFSVis

<table>
<thead>
<tr>
<th>ID</th>
<th>Context</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-VIS-08</td>
<td>Extensibility</td>
<td>The WFSVis must be extensible to handle new concepts (e.g., exact time simulations, new peripherals) in the future so that it is easy to add new peripherals into the WFS.</td>
<td>MUST</td>
</tr>
<tr>
<td>REQ-VIS-09</td>
<td>Portability</td>
<td>The WFSVis must have a connection with the TWINSCAN software running on a different platform.</td>
<td>MUST</td>
</tr>
<tr>
<td>REQ-VIS-10</td>
<td>Performance</td>
<td>When the WFSVis runs, the visualization should appear in less than ten seconds.</td>
<td>SHOULD</td>
</tr>
<tr>
<td>REQ-VIS-11</td>
<td>Performance</td>
<td>The WFSVis should not affect the performance of the WFS much that the total number of wafers produced per hour should not drop more than ten percent.</td>
<td>SHOULD</td>
</tr>
<tr>
<td>REQ-VIS-12</td>
<td>Response time</td>
<td>There is no strict real time requirement for the WFSVis but it should be responsive enough such that it should have more than twenty four frames per second (fps).</td>
<td>SHOULD</td>
</tr>
</tbody>
</table>

4.4 Design Criteria

While designing a system, application of a design criterion can improve the design. While designing the WFSVis, three design criteria are selected: impact, genericity, and documentation. In addition, two design criteria that do not apply (reliability and complexity) are given too. These criteria are revisited again in Section 8.1 to evaluate their effect on the design. In the following sections, each criterion is explained in detail.
4.4.1. Applicable Design Criteria

Impact

The WFSVis should be designed to have an impact on the debugging process of the wafer handler subsystem and the WFS. The main purpose of the WFSVis is to make the debugging easier and less time consuming for its users and this should be considered while making the design.

Genericity

There are different machine configurations which mean there are different simulators for each of those machines. The WFSVis should work with all of the different configurations. Therefore, while making the design, genericity criteria should be taken into account.

Documentation

The WFSVis is a user oriented tool. Its functional requirements are defined based on the user needs. It is important to get early feedback on the implemented features so that the design satisfies the user needs as much as possible. This can be achieved by developing prototypes incrementally with the new features. Therefore, the design should allow this type of iterative prototype development.

4.4.2. Non-applicable Design Criteria

Reliability

The reliability is not applicable to this project because it is known that the WFS simulator is a virtual concept of a 3D world and it can be mapped back to a 3D virtual world using a visualization engine. The challenge question is "How should it be visualized?" rather than "Can it be visualized?"

Complexity

The visualization of the WFS does not require any complex algorithms or complex structures. The visualizer is responsible for getting the correct data out of the WFS and showing it in 3D. It doesn't need to process any data coming from the simulator.
5. System Design

This chapter presents the architecture and the design of the WFSVis. To explain the architecture and the design, 4+1 architectural view is used which is explained in the next section. Then details of each view are given in subsequent sections. However, the process view, the development view and physical view are not included due to the corporate confidentiality.

5.1 4+1 Architectural View

The 4+1 Architectural View [7] describes the architecture of software systems, based on the use of multiple, concurrent views. In each view, different stakeholders are considered. In addition, a selection of use case scenarios describes the overall picture as 'plus one'. Hence the model contains 4+1 views:

- **Logical view**: The logical view focuses on realizing the functionality of the system in terms of structural elements, key abstractions and mechanism, separation of concerns and distribution of responsibilities. It is used for functional analysis and it contains class diagrams, package diagrams and state machine diagrams.

- **Process view**: The process view considers non-functional, dynamic aspects such as performance, scalability and throughput. It addresses the issues of concurrency, distribution and fault tolerance. The process view can be represented with sequence diagrams, activity diagrams and communication diagrams.

- **Development (implementation) view**: The development view focusses on configuration management and actual software module organization in the development environment and it is represented by component diagrams.

- **Physical (deployment) view**: The physical view encompasses the nodes that form the system’s hardware topology on which the system executes. It focusses on distribution, communication and provisioning and is represented by deployment diagrams.

- **Scenarios (use case view)**: In addition to the four views discussed above, this is the central view for capturing scenarios. It encompasses the use cases that describe the behavior of the system as seen by its end users and other stakeholders. It represents the scenarios that tie the previous four views together, and forms the reason why all the other views exist. It is represented by use case diagrams.

These five views are connected to each other (see Figure 5-1). The logical view will help the developer to extract the development and process views. Then these two view can be used together to come up with a physical view. During this process the scenarios can be used to see the big picture and make all these view consistent with each other.
5.2 Context of WFSVis

Before describing the use cases of WFSVis, its context is given in this section. TWINSCAN software runs on Solaris environment and sends the robot actuations to the simulator. Then, the simulator notifies the visualizer asynchronously. The visualizer component, structure this data and send it through TCP/IP channel to the 3D visualization engine runs on Windows. Then the 3D scene is updated accordingly (see Figure 5-2).
5.3 Use Case Scenarios

Use case scenarios show how external users interact with the WFSVis. They also describe the behavior of the system as seen by its end users and other stakeholders. In this section, the three main use case scenarios of the WFSVis that are extracted from the three domain problems described in Section 2.1 are given.

5.3.1. Visualizing the Wafer Handler Process

In this scenario, the user aims to observe the wafer handling process that runs with the TWINSCAN software. The user starts up the TWINSCAN software and establishes a connection with the WFSVis. Then the user registers the interested peripherals and materials to the visualizer. After that the position and the orientation of the selected peripherals are visible in the WFSVis screen. The peripherals that are actuated by the TWINSCAN (this actuation can be done manually as well) will be reflected in the 3D scene of the WFSVis and they will move to their destination positions accordingly. In Figure 5-3, an overview of this use case can be seen.

![Figure 5-3 Visualize Wafer Handler use case](image)

5.3.2. Visualizing Error Injections

In this scenario, the user aims to see the erroneous states injected to the WFS. First, the user adds the errors into the error configuration file with the respective injection format. Then user updates the errors in the WFS by calling a script file written for this purpose. Then the user connects the simulator to the visualizer. Next, the user runs the wafer handler process as usual. When the error injection occurs in the simulator, the 3D scenery is updated accordingly (e.g. the wafer will disappear in the case of a wafer lost error). The user will be warned that this scene change has occurred because of an error injection.
5.3.3. Visualizing a Trace File
In this scenario, the user aims to re-play and visualize previously produced trace (log) files. The user opens the visualizer and selects the trace file. This trace file is read by the WFSVis and events are re-played accordingly. The 3D scene will be updated accordingly.

5.4 Logical View
In this view, the logical layout of the WFSVis is given. Firstly, the structure for the WFS is given. Then the structure of the visualization and its commands that are used in the communication are given.

5.4.1. Wafer Flow Simulator
In the existing structure of the WFS, there is a controller class called System that possesses all the peripherals and wafers in the system (see Figure 5-6). The System class has modules which consist of different hardware elements (e.g., fixed, actuator, and sensor). There are different kinds of actuator classes to represent the diversity in the hardware (e.g., 1DOFZActuator, 3DOFActuator). In addition, there is a LightBeam-Sensor class to represent the nominal behavior shown in Figure 3-3.
5.4.2. Visualizer

The visualizer component is responsible for rendering the state of the WFS, which consists of peripheral actuations, peripheral states and the environment. It runs on Windows. It has a similar class structure to the simulator. However, instead of having directly two separate classes for actuators and sensor, it has a single common abstract class called VisualizerNode (see Figure 5-7). The actuator, fixed and sensor classes are derived from this class (see Figure 5-8). The detailed explanation about the VisualizerNode class is given in the next section.

Visualizer Node Structure

The node structure in the WFSVis is made to be consistent with the WFS node structure. The hierarchy of the nodes is structured in the same way. In Figure 5-8 you can see the structure in the WFSVis and compare it to the structure in the WFS by looking at Figure 5-6. The difference is in the class methods. The visualizer nodes have five different functions to either change their positions (move and rotate) or states (attach, detach, change). When an actuation is finished in the simulator (e.g. actuation on a robot with 3DOF), the corresponding function (in this case move) is called on the corresponding visualization node (in this case ThreeDOFActuator). The update function is used to update the position and/or state of each peripheral per frame. These functions that are used to control visualization nodes also form the basis of the command structure in the visualizer (see the next section).
Figure 5-7 Visualizer Structure

Visualizer Command Pattern

In the previous section, it is mentioned that the visualizer uses commands to change and update the 3D scene. In this section, detailed information about these commands is given.

In the visualizer node classes there are five different functions to update the 3D scene: attach, change, detach, move, and rotate (see Figure 5-8). Therefore, five different command classes are created to execute these functions:

- AttachCommand
- ChangeCommand
- DetachCommand
- MoveCommand
• RotateCommand

These command classes are encapsulated with an abstract Command class (see Figure 5-9). Each command class has private fields to hold the data needed to execute that function. For example, in MoveCommand there are three fields: x, y, and z. These values are used in move(double, double, double) function of a particular node. With this approach, each command class can have its own private fields that are needed and we do not need to create a common command class which holds all the required data together. This would decrease the network efficiency since each time we send only one command through network and the command object that is being sent should not have any redundant data.

Figure 5-9 Visualizer Command Structure

In order to execute a command, the execute function is called. Encapsulation of the execute function enables command objects to be executed on another platform. A command object can be instantiated on a platform, then can be sent over the network, and then can be executed on the remote platform using the execute function.

Execution of a command takes place on the command implementation which hides the implementation details of how each command affects a particular peripheral. The move function call on a robot and a peripheral is different in the WFSVis (see the note in Figure 5-9). This is further explained in the next section.

Rationale: There are several rationales behind the selection of the command pattern in the WFSVIS:

- Encapsulate a request (function call) as an object so that it can be serialized and sent through the network. (portability)
- Enable queuing or logging of the requests
- Decouple the WFSVis from the client (the WFS) so that the request can be done remotely since the creation of the commands does not depend on the WFSVis (remote object invocation). (portability)
- Make it extendable so that addition of new commands to the system will not require a lot of change in the system. (extendibility)
- The WFSVis can be tested using stub objects of the commands. (testability)

These commands are executed with a common interface function called execute. The execution of the commands depends on

- The existing nodes in the visualizer at the run time since they need to be altered in some way (either move, attach/detach or state change)

The factory to create new nodes when there is a creation command
These two dependencies are injected at runtime via execute function.

An explanation for each command is given below:

- **Attach Command**: This is used to attach a wafer to a peripheral.
- **Change Command**: This command is used to change states/values of sensors.
- **Create Command**: This command is used to create any kind of visualization node in the WFSVis.
- **Detach Command**: This is used to detach a wafer from a peripheral.
- **Move Command**: This command is used to move actuators.
- **Inject Command**: This command is used to indicate fault injections.

**Command Implementation Structure**

For each command in the WFSVis there is a common implementation interface (CommandImp, note the structural similarity between Figure 5-9 and Figure 5-10). This interface is implemented similarly to the visualizer node structure. By doing this, some common behaviors (attaching of the fixed or actuator peripherals, which has no effect) can be captured at higher levels of the tree.

**Rationale:**

- Decouple the commands from their implementations so that these two can vary independently from each other. In other words, an addition of a new command to the WFSVis will not change the structure of the command implementation and vice versa. (decoupling)
- Make it extendable so that addition of new peripherals to the system will not require a lot of change in the system. (extendibility)
- Hide the implementation details from the client (the WFS) so that these commands can be executed at some other location in the network. (portability)

For simplicity, only the implementations of the selected nodes are shown in Figure 5-10.
Bridge Pattern

The bridge pattern is used in software engineering to decouple an abstraction from its implementation so that the two can vary independently (see Figure 5-11). In the case of WFSVis project, the abstraction is the visualization commands (e.g., attach, move, create) and the implementation is how these operations are applied on each visualization node (e.g., 3DOF actuator node, 1DOFPhi actuator node).

Together with the command structure, the command implementation structure completes a bridge pattern. The abstraction part is covered by the command structure and the implementor part is covered by the command implementation structure. Using bridge pattern enables us to change or addition of new commands will result in minimal change in the implementation structure: the new command will be added as a function to the implementation classes.

Rationale:

- Decouple the commands from their implementations so that these two can vary independently from each other. In other words, an addition of a new command to the WFSVis will not change the structure of the command implementation and vice versa. (decoupling)
- Make it extendable so that addition of new commands and peripherals to the system will not require a lot of change. (extendibility)
- Hide the implementation details from the client (the WFS) so that these commands can be executed at some other location in the network. (portability)

![Figure 5-11 The Bridge Pattern]
6. Project Management

The tasks in this project are divided up into small parts using Work Breakdown Structure (WBS) technique. In the following section this structure is discussed in detail. Next, project milestones are defined and scheduling is done according to WBS. Figure 6-1 shows an overview of the milestones and their deadlines.

![Milestone deadlines diagram](image)

**Figure 6-1 Milestone deadlines**

### 6.1 Work Breakdown Structure

In this section, Work Breakdown Structure of the WFSVis project is discussed. The project is divided up to 5 major activities in the first level. These are: Planning and Management, Documents, Visualization Engine, Visualization Plugins, and Closeout. Each of these activities is further divided into smaller tasks. A more detailed task division can be seen in Figure 6-2.
Figure 6-2 Work Breakdown Structure of the WFSVis
### 6.2 Milestones

In this section, milestones and deliverables for each milestone are defined. The scheduling of the milestones is done according to tasks they contain. In Table 11, all the milestones and their subtasks can be seen.

**Table 11 Milestones and Task Distribution**

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Activities</th>
<th>Duration (weeks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem &amp; Domain Analysis (4 weeks)</td>
<td>1.1.1 Product Vision</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.3.1 Analyze Existing WFS</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.4.1 Examine Existing Log/Trace/Dump Files</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>4</strong></td>
</tr>
<tr>
<td>Requirement Analysis &amp; Project Plan (2 weeks)</td>
<td>1.1.2 Concept Matrix</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>1.1.3 Project Plan</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>1.1.4 Risk Management</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>1.2.1 Submit Requirements Document</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>2</strong></td>
</tr>
<tr>
<td>Design Phase &amp; Connection of Visualizer with WFS (4.5 weeks)</td>
<td>1.3.2 Verify User Requirements for Engine</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.3.3 Design Visualization Engine</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.4.2 Verify User Requirements for Plugins</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.4.3 Design Plugins</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2.2 Submit Design Document</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>4.5</strong></td>
</tr>
<tr>
<td>Detailed Wafer Flow Visualization (7.5 weeks)</td>
<td>1.3.4 Implement Visualization Engine</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1.3.5 Test Visualization Engine</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>1.3.6 Integrate Engine with WFS</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>7.5</strong></td>
</tr>
<tr>
<td>Plugins for Visualizer (6 weeks)</td>
<td>1.4.4 Implement Plugins</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1.4.5 Test Plugins</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.4.6 Integrate Plugins with WFS and Visualizer</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>6</strong></td>
</tr>
<tr>
<td>Documentation Phase (5 weeks)</td>
<td>1.2.3 Submit Project Visualizer Roadmap</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>1.2.4 Submit Change Case Scenarios</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>1.2.5 Submit Final Technical Report</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>5</strong></td>
</tr>
<tr>
<td>Final Presentation (4 weeks)</td>
<td>1.2.6 Submit Final Presentation</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.5.1 Final Presentation and Demonstration</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.5.2 Document Lessons Learned</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>1.5.3 Gain Formal Acceptance</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>4</strong></td>
</tr>
<tr>
<td>Buffer</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Grand Total</td>
<td></td>
<td><strong>36</strong></td>
</tr>
</tbody>
</table>
6.3 Risk Management

In this section, the risks involved with the WFSVis project are discussed. For each risk item, its impact and probability is defined within the scale from one to ten. In addition, a mitigation strategy is given. Whole list of the risk items can be seen in Table 12 and the overview of impact-probability analysis can be seen in Figure 6-3.

Table 12 The risk items involved with the WFSVIS project

<table>
<thead>
<tr>
<th>Risk Id</th>
<th>Risk Type</th>
<th>Description</th>
<th>Impact (1-10)</th>
<th>Prob. (1-10)</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSK-01</td>
<td>Process</td>
<td>Supervisor and/or domain experts are not available. (Early in the project)</td>
<td>8</td>
<td>4</td>
<td>Try to gain as much domain knowledge in the early steps. If it is blocking, focus on other project related tasks in time being.</td>
</tr>
<tr>
<td>RSK-02</td>
<td>Process</td>
<td>Supervisor and/or domain experts are not available. (Late in the project)</td>
<td>2</td>
<td>4</td>
<td>With the gained domain knowledge during the early phase of the project the impact is lowered.</td>
</tr>
<tr>
<td>RSK-03</td>
<td>Process</td>
<td>Illness. Trainee is ill for a long duration.</td>
<td>8</td>
<td>3</td>
<td>Keep a buffer time at the end of the project and negotiate requirements.</td>
</tr>
<tr>
<td>RSK-04</td>
<td>Process</td>
<td>Required tools for the development have not been delivered in time.</td>
<td>10</td>
<td>5</td>
<td>Signal to supervisor that project is blocked. While the problem is being solved focus on other project related tasks.</td>
</tr>
<tr>
<td>RSK-05</td>
<td>Technical</td>
<td>Changes will occur in the WFS during the project.</td>
<td>10</td>
<td>10</td>
<td>Create a branch of the WFS project in the code repository and once a month merge the branch with original WFS project to get the latest changes.</td>
</tr>
<tr>
<td>RSK-06</td>
<td>Technical</td>
<td>Lack of domain knowledge and lack of experience with ASML’s development tools</td>
<td>6</td>
<td>3</td>
<td>Read the documentation of WFS and ask questions to supervisors when got stuck.</td>
</tr>
<tr>
<td>RSK-07</td>
<td>Technical</td>
<td>Firewall on development PC might block the network communication with Solaris machine.</td>
<td>8</td>
<td>8</td>
<td>Early test of network communication. If the communication is blocked by firewall, request network permission.</td>
</tr>
<tr>
<td>RSK-08</td>
<td>Technical</td>
<td>The software structure is complex.</td>
<td>4</td>
<td>5</td>
<td>Push the integration phase to earlier</td>
</tr>
</tbody>
</table>
### Technical

| RSK-09 | Lack of knowledge about Solaris operating system might cause problems during the deployment phase. | 7 | 7 | Try to develop one of the early prototypes to Solaris to detect possible problems as early as possible so that there is enough time to solve them. |

### Figure 6-3 Risk impact vs. risk probability chart

![Risk Impact vs Probability Chart](image-url)
7. Conclusion and Future Directions

In this chapter, concluding remarks about the WFSVis project is given in the next section. In Section 7.2, the limitation of the current version is explained. In Section 7.3, the possible future features are discussed.

7.1 Conclusions

This project proposes a visualization tool for the wafer handler subsystem. The visualization tool is based on the WFS and has the following key deliverables:

- **Visualization Software**: With the visualization software (on Windows), the user able to capture the internal states of the peripherals, the wafers, and the sensors in the WFS. Having all the information in a 3D scene is valuable in terms of understanding how the wafer handler works. The visual information provided by the WFSVis can be also used to debug the WFS functionality. Moreover, the WFSVis can also generate 3D scenes for system configuration that are still under design, by using their model specifications and this 3D scene gives information about the system to the developers who haven't seen its layout.

- **Visualization Code Generator**: The code generator is extended to produce visualization functionality. With this feature, a visualization tool can be generated for any of the system configurations. This is important for ASML since there are different project with different configurations running in parallel.

- **Log File Parser**: The parser tool uses the log files are produced by the TWINSCAN software. The offline visualization of the log files makes debugging easier since the user can follow the actuations in a 3D scene instead of going through thousands lines of text.

The WFSVis already used in developing and executing new test cases for the WFS. It also received good feedback from the developers of the QXE system which was still under design at the time this project was developed. The QXE system uses a wafer handler subsystem with different layout and the visualization tool was able to show it in a 3D scene using its model file.

The main challenge in this project was to design an adaptable system to WFS changes. The visualization tool first designed for sequential actuations of the peripherals. In the later steps of the project, the design is adapted easily to handle parallel actuations of the robot with actual timings. Another challenge was to design it as a portable system that is going to be deployed on a platform, which is Solaris that I have no experience with. The last challenge was to design a generic system. WFSVis is generic to handle different system configurations (e.g., NXE, QXE).

7.2 Limitations

The limitation of the WFSVis is that the users should have a port access permission in their PC firewall settings. This is due to the network infrastructure of ASML and the deployment environment. Each development PC has a pre-configured firewall installed to block the entire incoming connection request for safety. This includes the port that the WFSVis uses.
7.3 **Future Direction**

The visualization tool has a lot of potential to be further developed. The functionalities for future development are explained in the following sections.

7.3.1. **Import AutoCAD Models**
The meshes that are used in the WFSVis are handmade. They are created based on the hardware drawings. It would be better if the actual AutoCAD models for the hardware parts are imported and used as meshes. With this feature, better realism of the machine layout can be achieved.

7.3.2. **Collision Detection**
In the current version, the visualizer does not check for possible collisions in the system because the meshes are manually created and the details in their shapes are unknown. However, once the actual AutoCAD models are imported and used within the tool, the required position information for all the peripherals will be available to the WFSVis. Then, this information can be used to develop a collision detection feature.

7.3.3. **Benchmark**
There is no benchmarking done for the current version of the WFSVis. A benchmark on the command execution delay (the time difference between the actuation in the WFS and its command execution in the WFSVis), the rendering performance (the frame rate per second based on the number of peripherals in the system), and the delay caused in the WFS execution time.

7.3.4. **More Robust Network Protocol**
The current network protocol of the WFSVis uses (de)serialization to send the commands over the network. However, there is no synchronization or hand shake protocol implemented. It would be better to use an existing network protocol such as Simple Object Access Protocol (SOAP) to prevent data loss and increase the robustness of the WFS.
8. Project Retrospective

In this chapter, the design criteria that are mentioned in Section 4.4 are revisited. Moreover, the experience that I collect during the project is explained.

8.1 Design Criteria Revisited

In this section, how the chosen design criteria affected/improved the design is discussed. The three design criteria that are selected in Section 4.4 were impact, genericity, and documentation.

8.1.1. Impact
The WFSVis tries to make debugging more efficient and understanding of the wafer handler subsystem easier. During its design phase, these impacts on the development process are considered. Therefore, the visualization system is designed to reflect events and actuations happening in the WFS to its users as close as possible to real time.

Having an impact on the development process requires an easily deployable system. Therefore, the WFSVis is designed in such a way that the user spends minimum effort to deploy it and get it working.

8.1.2. Genericity
The second important design aspect is genericity. Thinking about a generic system during the design phase led to the extension of the existing code generator with the visualization features. The WFSVis designed to work any VPDSL model file. The advantage of this was seen when a nonexistent system (QXE) is visualized with the WFSVis and this proved.

8.1.3. Documentation
The iterative development of the tool and getting regular feedback weekly resulted in an extensible product. The WFSVis is designed to be easily extended. In the first weeks, a small prototype that can only visualize robot is implemented. Then, iteratively, the other peripherals are added into the prototype. With this approach, getting early feedback affected the development process positively. The mistakes are corrected way before they become too costly to make a change in the design.

8.2 Experience

During the project, I have come across several challenges. The first challenge was to design the visualization to be deployed on the target platform (Solaris) and integrate the visualizer with the TWINSCAN software. Solaris and the development environment of ASML were not my expertise when I started the project. Therefore, this integration was the most risky issue. We decided to push the integration as early as possible in the project (in the third month milestone of the project, the integration was done) to have enough time to solve all the problems encountered during the process. At the end, this turned to be a really good idea. When I was integrating the visualizer, the progress speed went a bit down compared to other milestone, but when it is finished the risk was not there anymore and we had an integrated, working prototype.

I also learned a lot of information about the development environment of ASML. If I had this experience before, I would do my initial design differently. This experience is also valuable for my future career since after this project I will pursue my career in ASML. The tool-set in ASML has a steep learning curve and the experience I had in the project will help me to adapt the new department I am going to work at.
Glossary

**domain-specific language**: A domain-specific language (DSL) is a type of programming language or specification language in software development and domain engineering dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique.

**lithography**: The method for printing using a stone (lithographic limestone) or a metal plate onto a paper.

**mask**: A carrier with the negative or positive version of a (part of a) pattern of a layer of a microlithography device.

**mesh**: A polygon mesh is a collection of vertices, edges and faces that defines the shape of a polyhedral object in 3D computer graphics and solid modeling.

**model-driven engineering**: Model-driven engineering (MDE) is a software development methodology which focuses on creating and exploiting domain models (that is, abstract representations of the knowledge and activities that govern a particular application domain), rather than on the computing (or algorithmic) concepts.

**photolithography**: The process used in microchip production to imprints a pattern on a substrate using light.

**reticle**: A carrier with the negative or positive version of a (part of a) pattern of a layer of a microlithography device.

**reticle stage**: The subsystem in the photolithography machine that is used to position a reticle during an exposure.

**throughput**: The amount of material or items passing through a system or process. In this context, it is the number of wafers imprinted successfully in one hour.

**TWINSCAN**: The photolithography machine that ASML is producing.

**wafer**: Circular slice of semi-conducting material that serves as a substrate for the manufacturing of chips.

**wafer handler**: The subsystem in the photolithography machine that is responsible of aligning the wafer before it is moved to wafer stage subsystem.

**wafer stage**: The subsystem in the photolithography machine that is responsible of positioning wafer during the exposure process.
Bibliography


About the Author

Umut Uyumaz received his B.Sc. degree in Computer Engineering in May 2009 from Bilkent University, Turkey. Then he received his M.Sc. degree in Computer Science in September 2011 from Swiss Federal Institute of Technology (ETH) Zurich. In his bachelor thesis, he worked on analyzing customer attention to display windows using a web cam. In his master thesis, he proposed a method on 3D modeling with symmetric sketches. His main research interests are 3D Modeling and Computer Vision.
3TU. School for Technological Design, Stan Ackermans Institute offers two-year postgraduate technological designer programmes. This institute is a joint initiative of the three technological universities of the Netherlands: Delft University of Technology, Eindhoven University of Technology and University of Twente. For more information please visit: www.3tu.nl/sai.